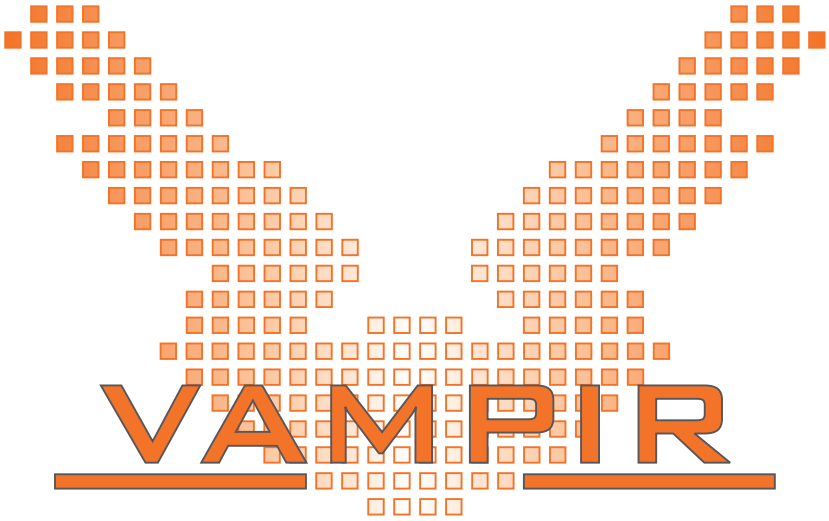
GWT forschung+innovation

Vampir 8

User Manual



Copyright

© 2013 GWT-TUD GmbH

Blasewitzer Str. 43
01307 Dresden, Germany

http://gwtonline.de

Support / Feedback / Bugreports

Please provide us feedback! We are very interested to hear what people like, dislike, or what features they are interested in.

If you experience problems or have suggestions about this application or manual, please contact service@vampir.eu.

When reporting a bug, please include as much detail as possible in order to reproduce it. Please send the version number of your copy of Vampir along with the bug report. The version is stated in the *About Vampir* dialog accessible from the main menu under *Help* → *About Vampir*.

Please visit http://vampir.eu for updates.

service@vampir.eu
http://vampir.eu

Manual Version

2013-06 / Vampir 8.1

Contents

1 Introduction **5**
1.1 Event-based Performance Tracing and Profiling . . . 5
1.2 The Open Trace Format (OTF) . . . 6
1.3 Vampir and Windows HPC Server 2008 . . . 7

2 Getting Started **8**
2.1 Installation of Vampir . . . 8
2.1.1 Linux, Unix . . . 8
2.1.2 Mac OS X . . . 8
2.1.3 Windows . . . 9
2.2 Generation of Performance Data . . . 9
2.2.1 Score-P . . . 9
2.2.2 VampirTrace . . . 11
2.2.3 Event Tracing for Windows (ETW) . . . 11
2.3 Starting Vampir and Loading Performance Data . . . 13
2.3.1 Loading a Trace File . . . 14
2.3.2 Loading a Trace File Subset . . . 15

3 Basics **17**
3.1 Chart Arrangement . . . 18
3.2 Context Menus . . . 20
3.3 Zooming . . . 22
3.4 The Zoom Toolbar . . . 24
3.5 The Charts Toolbar . . . 25
3.6 Properties of the Trace File . . . 26

4 Performance Data Visualization **27**
4.1 Timeline Charts . . . 27
4.1.1 Master Timeline and Process Timeline . . . 27
4.1.2 Counter Data Timeline . . . 32
4.1.3 Performance Radar . . . 34
4.2 Statistical Charts . . . 41
4.2.1 Function Summary . . . 41
4.2.2 Process Summary . . . 43
4.2.3 Message Summary . . . 44
4.2.4 Communication Matrix View . . . 45

4.2.5 I/O Summary . . . 46
4.2.6 Call Tree . . . 47
4.3 Informational Charts . . . 48
4.3.1 Function Legend . . . 48
4.3.2 Marker View . . . 48
4.3.3 Context View . . . 49
4.4 Customizable Performance Metrics . . . 52
4.4.1 Metric Editor . . . 53
4.4.2 Examples . . . 55

5 Information Filtering and Reduction 59
5.1 Process Filter . . . 60
5.2 Message and Collective Operations Filter . . . 61
5.3 I/O Filter . . . 62
5.4 Function Filter . . . 63
5.4.1 Filter Options . . . 64
5.4.2 Examples . . . 67

6 Comparison of Trace Files 77
6.1 Starting and Saving a Comparison Session . . . 78
6.2 Usage of Charts . . . 80
6.3 Alignment of Multiple Trace Files . . . 82
6.4 Usage of Predefined Markers . . . 84

7 Customization 86
7.1 General Preferences . . . 86
7.2 Appearance . . . 87
7.3 Saving Policy . . . 88

8 A Use Case 90
8.1 Introduction . . . 90
8.2 Identified Problems and Solutions . . . 91
8.2.1 Computational Imbalance . . . 91
8.2.2 Serial Optimization . . . 93
8.2.3 High Cache Miss Rate . . . 94
8.3 Conclusion . . . 96

1 Introduction

Performance optimization is a key issue for the development of efficient parallel software applications. *Vampir* provides a manageable framework for analysis, which enables developers to quickly display program behavior at any level of detail. Detailed performance data obtained from a parallel program execution can be analyzed with a collection of different performance views. Intuitive navigation and zooming are the key features of the tool, which help to quickly identify inefficient or faulty parts of a program code. Vampir implements optimized event analysis algorithms and customizable displays which enable a fast and interactive rendering of very complex performance monitoring data. Ultra large data volumes can be analyzed with a parallel version of Vampir, which is available on request.

Vampir has a product history of more than 15 years and is well established on Unix based HPC systems. This tool experience is also available for HPC systems that are based on *Microsoft Windows HPC Server 2008*.

1.1 Event-based Performance Tracing and Profiling

In software analysis, the term profiling refers to the creation of tables, which summarize the runtime behavior of programs by means of accumulated performance measurements. Its simplest variant lists all program functions in combination with the number of invocations and the time that was consumed. This type of profiling is also called inclusive profiling, as the time spent in subroutines is included in the statistics computation.

A commonly applied method for analyzing details of parallel program runs is to record so-called trace log files during runtime. The data collection process itself is also referred to as tracing a program. Unlike profiling, the tracing approach records timed application events like function calls and message communication as a combination of timestamp, event type, and event specific data. This creates a stream of events, which allows very detailed observations of parallel programs. With this technology, synchronization and communication patterns of parallel program runs can be traced and analyzed in terms of performance and correctness. The analysis is usually carried out in a postmortem step, i.e., after completion of the program. It is needless to say

that program traces can also be used to calculate the profiles mentioned above. Computing profiles from trace data allows arbitrary time intervals and process groups to be specified. This is in contrast to profiles accumulated during runtime.

1.2 The Open Trace Format (OTF)

The *Open Trace Format* (OTF) was designed as a well-defined trace format with open, public domain libraries for writing and reading. This open specification of the trace information provides analysis and visualization tools like Vampir to operate efficiently at large scale. The format addresses large applications written in an arbitrary combination of Fortran77, Fortran (90/95/etc.), C, and C++.

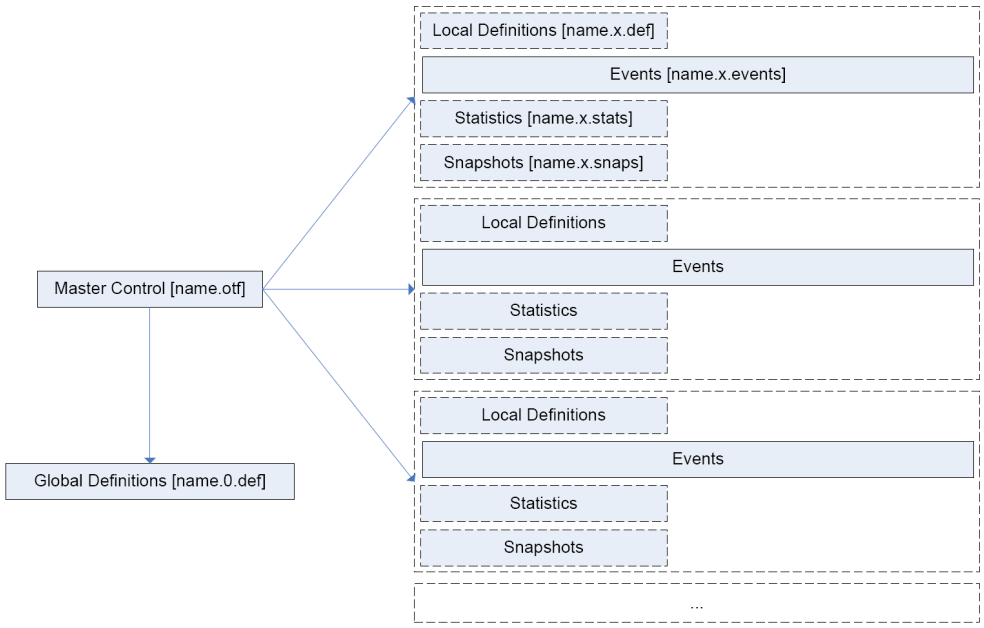


Figure 1.1: Representation of Streams by Multiple Files

OTF uses a special ASCII data representation to encode its data items with numbers and tokens in hexadecimal code without special prefixes. That enables a very powerful format with respect to storage size, human readability, and search capabilities on timed event records.

In order to support fast and selective access to large amounts of performance trace data, OTF is based on a stream-model, i.e. single separate units representing segments of the overall data. OTF streams may contain multiple independent processes whereas a process belongs to a single stream exclusively. As shown in Figure 1.1, each stream is represented by multiple files which store definition records, performance

events, status information, and event summaries separately. A single global master file holds the necessary information for the process to stream mappings.

Each file name starts with an arbitrary common prefix defined by the user. The master file is always named {*name*}*.otf*. The global definition file is named {*name*}*.0.def*. Events and local definitions are placed in files {*name*}*.x.events* and {*name*}*.x.defs* where the latter files are optional. Snapshots and statistics are placed in files named {*name*}*.x.snaps* and {*name*}*.x.stats* which are optional, too.

Note: Open the master file (**.otf*) to load a trace. When copying, moving or deleting traces it is important to take all according files into account otherwise Vampir will render the whole trace invalid! Good practice is to hold all files belonging to one trace in a dedicated directory.

Detailed information about the Open Trace Format can be found in the Open Trace Format (OTF)[1] documentation.

1.3 Vampir and Windows HPC Server 2008

The Vampir performance visualization tool usually consists of a performance monitor (e.g., Score-P, see Section 2.2.1 or VampirTrace, see Section 2.2.2) that records performance data and a performance GUI, which is responsible for the graphical representation of the data. In Windows HPC Server 2008, the performance monitor is fully integrated into the operating system, which simplifies its employment and provides access to a wide range of system metrics. A simple execution flag controls the generation of performance data. This is very convenient and an important difference to solutions based on explicit source, object, or binary modifications. Windows HPC Server 2008 is shipped with a translator, which produces trace log files in Vampir's Open Trace Format (OTF). The resulting files can be visualized with the Vampir performance data browser.

[1] http://www.tu-dresden.de/zih/otf

2 Getting Started

2.1 Installation of Vampir

Vampir is available for all major platforms. Its installation process depends on the target operation system. The following sections explain the particular installation steps for each system.

2.1.1 Linux, Unix

An installer package is provided for Linux/Unix systems. To install Vampir run the installer from the command line.

```
./vampir-8.1.0-standard-linux-ia32-setup.bin
```

Additional instructions are provided during installation. For an overview of all available options run the installer package with the option `--help`.

It is possible to run the installer in silent (unattended) mode with the `-s` command line option. In this case the installer assumes default values for all options.

By default, the installer associates Vampir with OTF and OTF2 files (**.otf, *.otf2*). This allows to quickly open a trace file by double-clicking its master file. Furthermore, a desktop icon and a desktop dependent menu items are generated.

During the first start of Vampir the license installation is completed.

Finally, Vampir can be launched via the respective desktop icon or by using the command line interface (see Section 2.3).

2.1.2 Mac OS X

Open the .dmg installation package and drag the Vampir icon into the applications folder on your computer. You might need administrator rights to do so. Alternatively, you can also drag the Vampir application to another directory that is writable for you. After that, double click on the Vampir application and follow the instructions for license installation.

2.1.3 Windows

On Windows platforms the provided Vampir installer makes the installation very simple and straightforward. Just run the installer and follow the installation wizard. Install Vampir in a folder of your choice, e.g.:

```
C:\Program Files
```

In order to run the installer in silent (unattended) mode use the `/S` option. It is also possible to specify the output folder of the installation with `/D=dir`. An example of a silent installation command is as follows:

```
Vampir-8.1.0-Standard-x86-setup.exe /S /D=C:\Program Files
```

You also have the option to associate Vampir with OTF and OTF2 files (**.otf, *.otf2*) during the installation process. This allows you to load a trace file quickly by double-clicking its master file. Subsequently, Vampir can be launched by double-clicking its icon or by using the command line interface (see Chapter 2.3).

At the first start Vampir will display instructions for license installation.

2.2 Generation of Performance Data

The generation of trace log files for the Vampir performance visualization tool requires a working monitoring system to be attached to your parallel program. The following software packages provide compatible monitoring systems with built-in support for the Vampir performance data file format.

2.2.1 Score-P

Score-P is the recommended code instrumentation and run-time measurement framework for Vampir 8. The goal of Score-P is to simplify the analysis of the behavior of high performance computing software and to allow the developers of such software to find out where and why performance problems arise, where bottlenecks may be expected and where their codes offer room for further improvements with respect to the run time. A number of tools have been around to help in this respect, but typically each of these tools has only handled a certain subset of the questions of interest. A crucial problem in the traditional approach used to be the fact that each analysis tool had its own instrumentation system, so the user was commonly forced to repeat the instrumentation procedure if more than one tool was to be employed. In this context, Score-P offers the user a maximum of convenience by providing the Opari2 instrumentor as a common infrastructure for a number of analysis tools like Periscope, Scalasca, Vampir, and Tau

that obviates the need for multiple repetitions of the instrumentation and thus substantially reduces the amount of work required. It is open for other tools as well. Moreover, Score-P provides the new Open Trace Format Version 2 (OTF2) for the tracing data and the new CUBE4 profiling data format which allow a better scaling of the tools with respect to both the run time of the process to be analyzed and the number of cores to be used. Score-P supports the programming paradigms serial, OpenMP, MPI, and hybrid (MPI combined with OpenMP).

Internally, the instrumentation itself will insert special measurement calls into the application code at specific important points (events). This can be done in an almost automatic way using corresponding features of typical compilers, but also semi-automatically or in a fully manual way, thus giving the user complete control of the process. In general, an automatic instrumentation is most convenient for the user. This is done by using the `scorep` command that needs to be prefixed to all the compile and link commands usually employed to build the application. Thus, an application executable app that is normally generated from the two source files *app1.f90* and *app2.f90* via the command:

```
mpif90 app1.f90 app2.f90 -o app
```

will now be built by:

```
scorep mpif90 app1.f90 app2.f90 -o app
```

using the Score-P instrumentor.

When makefiles are employed to build the application, it is convenient to define a placeholder variable to indicate whether a preparation step like an instrumentation is desired or only the pure compilation and linking. For example, if this variable is called `PREP` then the lines defining the C compiler in the makefile can be changed from:

```
MPICC = mpicc
```

to

```
MPICC = $(PREP) mpicc
```

(and analogously for linkers and other compilers). One can then use the same makefile to either build an instrumented version with the

```
make PREP="scorep"
```

command or a fully optimized and not instrumented default build by simply using:

```
make
```

in the standard way, i.e. without specifying `PREP` on the command line.

Detailed information about the installation and usage of Score-P can be found in the Score-P user manual[1].

[1] http://www.score-p.org

2.2.2 VampirTrace

VampirTrace used to be the recommended monitoring facility for Vampir. It is still available as Open Source software but no longer under active development (see Score-P Section 2.2.1). During a program run of an application, VampirTrace generates an OTF trace file, which can be analyzed and visualized by Vampir. The VampirTrace library allows MPI communication events of a parallel program to be recorded in a trace file. Additionally, certain program-specific events can be included. To record MPI communication events, simply re-link the program with the VampirTrace library. A new compilation of the program source code is only necessary if program specific events should be added.

To perform measurements with VampirTrace, the application program needs to be instrumented, which is done automatically. All the necessary instrumentation steps are handled by the compiler wrappers of VampirTrace (vtcc, vtcxx, vtf77, vtf90 and the additional wrappers mpicc-vt, mpicxx-vt, mpif77-vt, and mpif90-vt in Open MPI 1.3). All compile and link commands in the used makefile should be replaced by the VampirTrace compiler wrapper, which performs the necessary instrumentation of the program and links the suitable VampirTrace library. Simply use the compiler wrappers without any parameters, e.g.:

```
vtf90 hello.f90 -o hello
```

Running a VampirTrace instrumented application results in an OTF trace file stored the current working directory where the application was executed. On Linux, Mac OS X, and Sun Solaris the default name of the trace file will be equal to the application name. For other systems, the default name is `a.otf` but can be defined manually by setting the environment variable `VT_FILE_PREFIX` to the desired name.

Detailed information about the installation and usage of VampirTrace can be found in the VampirTrace user manual[2].

2.2.3 Event Tracing for Windows (ETW)

The *Event Tracing for Windows* (ETW) infrastructure of the *Windows* client and server OS's provides a powerful software monitor. Starting with *Windows HPC Server 2008* MS-MPI has built-in support for this monitor. It enables application developers to quickly produce traces in production environments by simply adding an extra mpiexec flag (`-trace`). Trace files will be generated during the execution of your application. The recorded trace log files include the following events: Any MS-MPI application call and low-level communication within sockets, shared memory, and NetworkDirect implementations. Each event includes a high-precision CPU clock timer for precise visualization and analysis.

[2] http://www.tu-dresden.de/zih/vampirtrace

The steps necessary for monitoring the MPI performance of an MS-MPI application are depicted in Figure 2.1. First the application needs to be available throughout all compute nodes in the cluster and has to be started with tracing enabled. The Event Tracing for Windows (ETW) infrastructure writes event logs (*.etl* files) containing the respective MPI events of the application on each compute node. In order to achieve consistent event data across all compute nodes clock corrections need to be applied. This step is performed after the successful run of the application using the Microsoft tool `mpicsync`. Now the event log files can be converted into OTF files with help of the tool `etl2otf`. The last necessary step is to copy the generated OTF files from the compute nodes into one shared directory. Then this directory includes all files needed by Vampir. The application performance can be analyzed now.

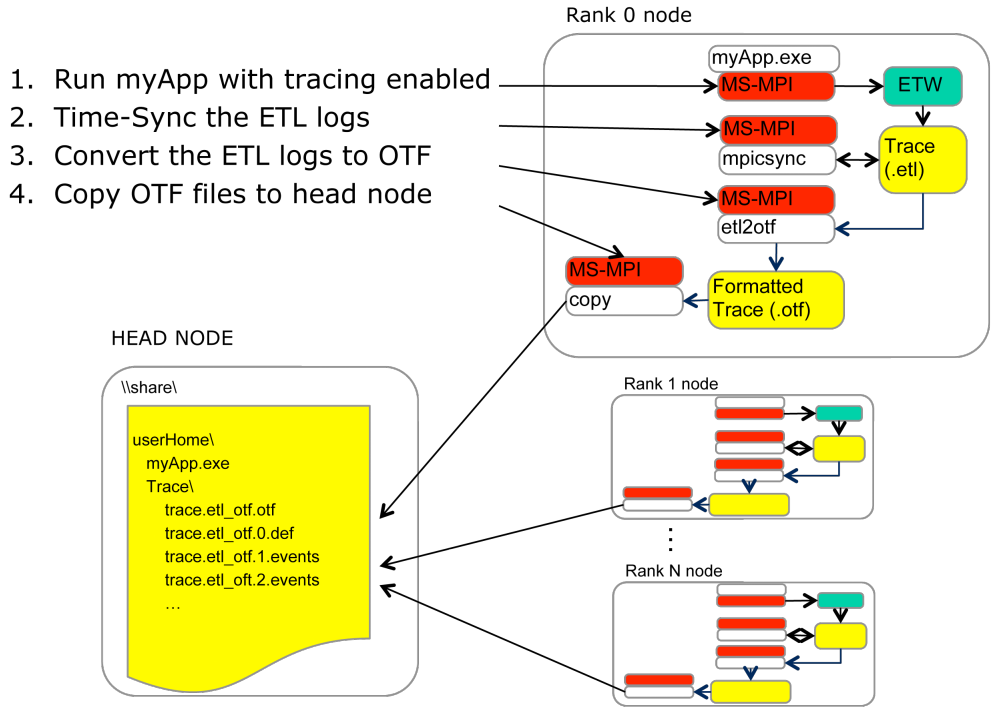


Figure 2.1: MS-MPI Tracing Overview

The following commands illustrate the procedure described above and show, as a practical example, how to trace an application on the Windows HPC Server 2008. For proper utilization and thus successful tracing, the file system of the cluster needs to meet the following prerequisites:

- `\\share\userHome` is the shared user directory throughout the cluster
- MS-MPI executable `myApp.exe` is available in the shared directory
- `\\share\userHome\Trace` is the directory where the OTF files are collected

1. Launch application with tracing enabled (use of `-tracefile` option):

```
mpiexec -wdir \\share\userHome\
-tracefile %USERPROFILE%\trace.etl myApp.exe
```

- `-wdir` sets the working directory; `myApp.exe` has to be there
- `%USERPROFILE%` translates to the local home directory, e.g. `C:\Users\userHome`; on each compute node the event log file (.etl) is stored locally in this directory

2. Time-sync the event log files throughout all compute nodes:

```
mpiexec -cores 1 -wdir %USERPROFILE% mpicsync trace.etl
```

- `-cores 1`: run only one instance of mpicsync on each compute node

3. Format the event log files to OTF files:

```
mpiexec -cores 1 -wdir %USERPROFILE% etl2otf trace.etl
```

4. Copy all OTF files from compute nodes to trace directory on share:

```
mpiexec -cores 1 -wdir %USERPROFILE% cmd /c copy /y
"*_otf*" "\\share\userHome\Trace"
```

2.3 Starting Vampir and Loading Performance Data

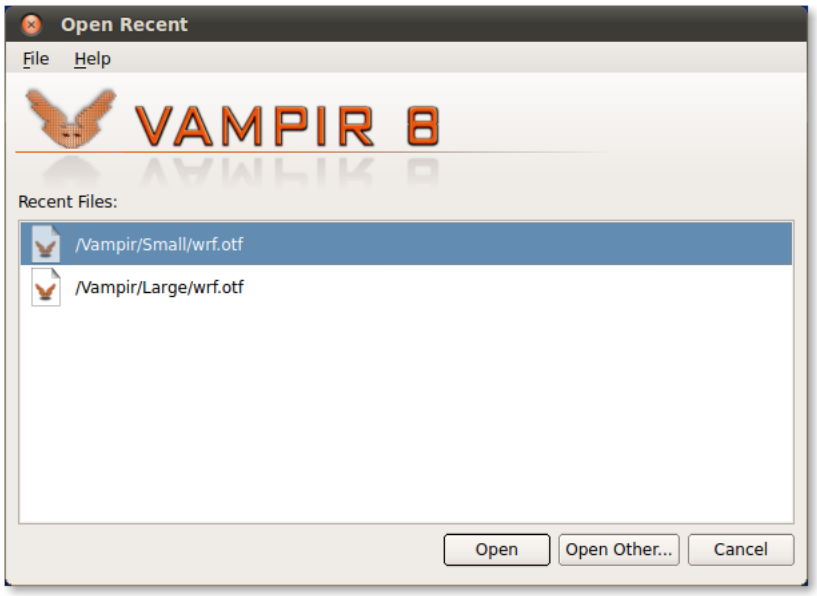


Figure 2.2: List of recent trace files

Viewing performance data with the Vampir GUI is very easy. On Windows the tool can be started by double clicking its desktop icon (if installed) or by using the Start Menu.

On a Linux-based machine run `./vampir` in the directory where Vampir is installed. A double click on the application icon opens Vampir on Mac OS X systems.

At startup Vampir presents a list of recently loaded trace files as depicted in Figure 2.2. Selecting a list entry and clicking the *Open* button loads the respective trace. The recent list is empty when Vampir is started for the first time.

2.3.1 Loading a Trace File

To open an arbitrary trace file, click on *Open Other...* or select *Open...* in the *File* menu, which provides the file open dialog depicted in Figure 2.3.

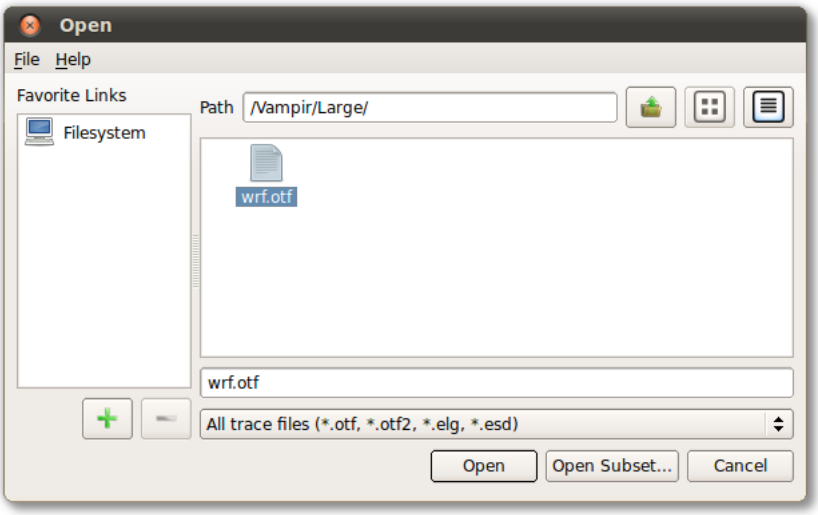


Figure 2.3: Loading a trace file in Vampir

It is possible to filter the files in the list. The file type input selector determines the visible files. The default is *All trace files (*.otf, *.otf2, *.elg, *.esd)*, which only shows trace files that can be processed by the tool. All file types can be displayed by using *All Files (*)*.

After selection of the trace file the loading process is started by a click on the *Open* button. Alternatively, a command line invocation is possible. The following command line sequence shows an example for a Windows system. Other platforms work accordingly.

```
C:\Program Files\Vampir\Vampir.exe [trace file]
```

To open multiple trace files at once you can give them one after another as command line arguments:

```
C:\Program Files\Vampir\Vampir.exe [file 1]...[file n]
```

If Vampir was associated with **.otf/*.otf2* files during the installation process, it is also possible to start the application by double-clicking an **.otf/*.otf2* file.

While Vampir is loading the trace file, an empty *Trace View* window with a progress bar at the bottom opens. After Vampir loaded the trace data completely, a default set of charts will appear. The loading process can be interrupted at any time by clicking the cancel button in the lower right corner of the Trace View. Because events in the trace file are loaded one after another, the GUI will open and show the earliest, already loaded information from the trace file.

The basic functionality and navigation elements of the GUI are described in Chapter 3. The available charts and the information provided by them are explained in Chapter 4.

2.3.2 Loading a Trace File Subset

To handle large trace files and save time and memory resources, it is possible to load only a performance data subset from a trace file. For this purpose the open dialog, Figure 2.3, provides the button *Open Subset...*. Clicking on this button opens a trace data pre-selection dialog as depicted in Figure 2.4.

An overview snapshot of the recorded application run is given at the top of the dialog. The time range of interest can be set with the edge markers on the left and right of the overview snapshot. Likewise, the time range to be loaded can be set explicitly in the input fields *From:* and *To:*. If markers are available in the trace file, their timing information can be used as reference points as well. Two markers need to be selected first (use shift + mouse click for the second marker). Next, click on *Zoom Between Marker* to set the respective time interval in the *From:* and *To:* input fields. The event data to be loaded can also be restricted to certain processes or threads of execution by disabling unwanted instances in the selection area entitled *Processes* (see Section 5.1 for further details). Once the data subset of interest is specified a click on the *OK* button starts the loading process.

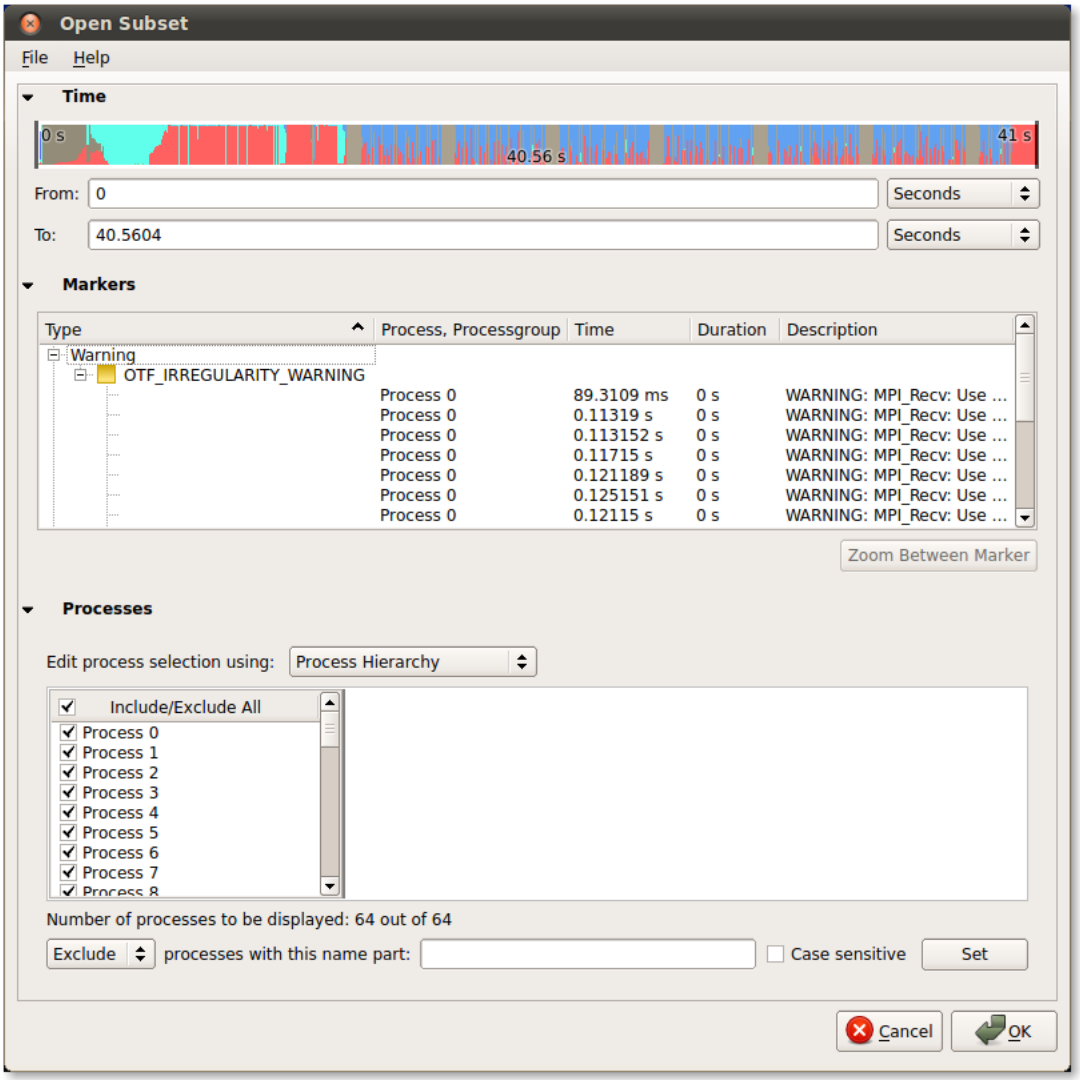


Figure 2.4: Selecting a trace data subset to be loaded

3 Basics

After loading has been completed, the *Trace View* window title displays the trace file's name as depicted in Figure 3.1. By default the *Charts* toolbar and the *Zoom Toolbar* are available.

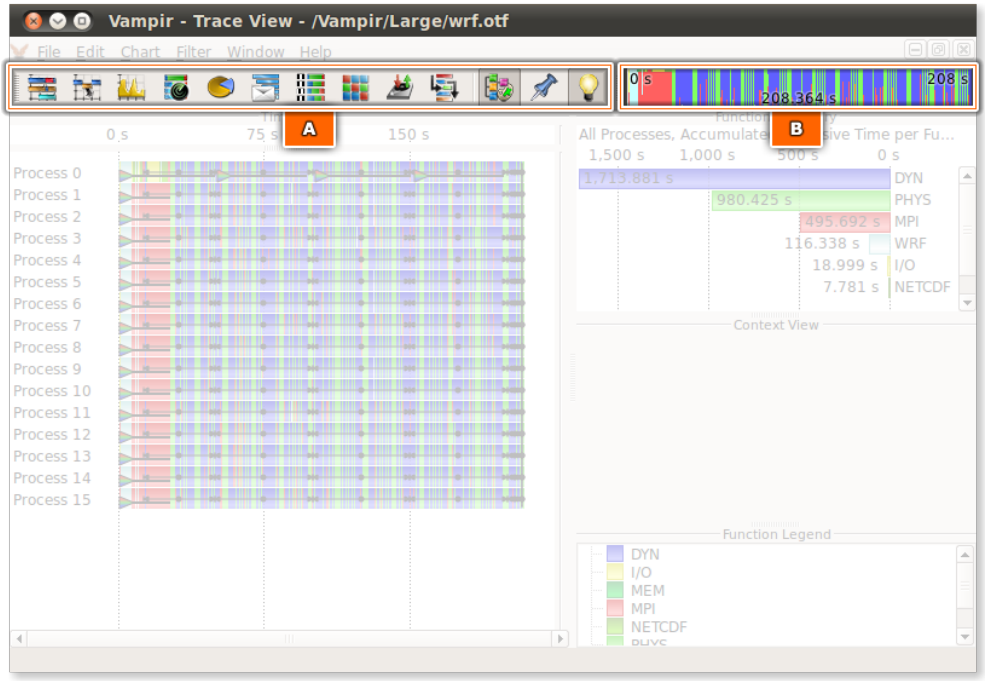


Figure 3.1: Trace View Window with Charts Toolbar (A) and Zoom Toolbar (B)

Furthermore, a default set of charts is opened automatically after loading has been finished. The charts can be divided into three groups: timeline-, statistical-, and informational charts. Timeline charts show detailed event based information for arbitrary time intervals while statistical charts reveal accumulated measures which were computed from the corresponding event data. Informational charts provide additional or explanatory information regarding timeline- and statistical charts. All available charts can be opened with the *Charts* toolbar which is explained in Chapter 3.5.

In the following sections we will explain the basic functions of the Vampir GUI which are generic to all charts. If you are already familiar with the fundamentals feel free to skip this chapter. The details of the different charts are explained in Chapter 4.

3.1 Chart Arrangement

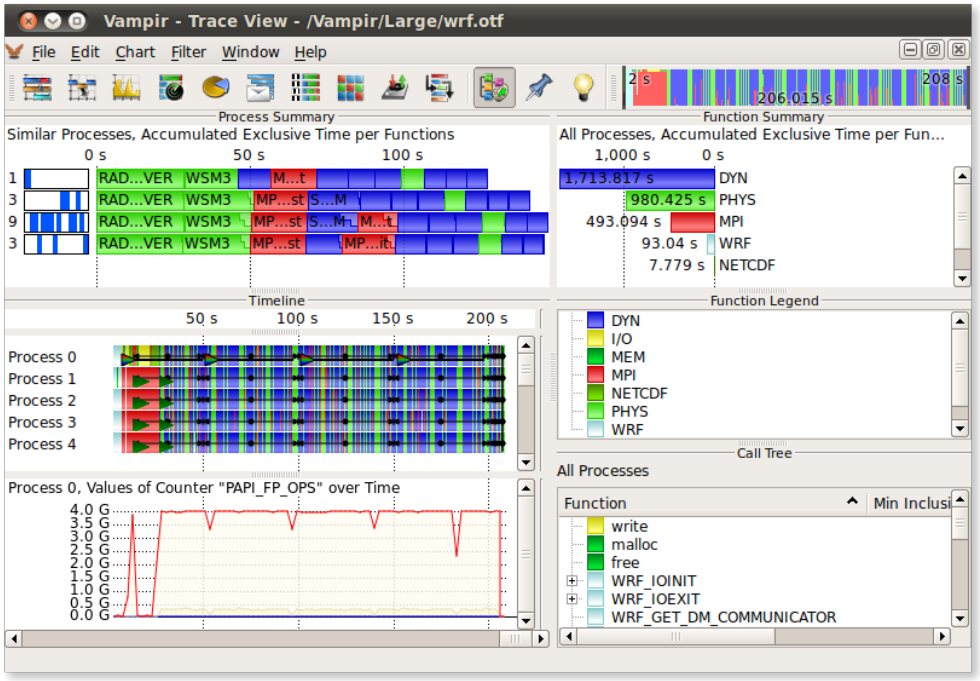


Figure 3.2: A Custom Chart Arrangement in the Trace View Window

The utility of charts can be increased by correlating them and their provided information. Vampir supports this mode of operation by allowing to display multiple charts at the same time. All timeline charts, such as the *Master Timeline* and the *Process Timeline* display a sequence of events. Those charts are therefore aligned vertically. This alignment ensures that the temporal relationship of events is preserved across chart boundaries.

The user can arrange the placement of the charts according to his preferences by dragging them into the desired position. When the left mouse button is pressed while the mouse pointer is located above a placement decoration, the layout engine will give visual clues as to where the chart may be moved. As soon as the user releases the left mouse button the chart arrangement will be changed according to his intentions. The entire procedure is depicted in Figures 3.3 and 3.4.

The layout engine furthermore allows a flexible adjustment of the screen space that is used by a chart. Charts of particular interest may get more space in order to render information in more detail.

The *Trace View* window can host an arbitrary number of charts. Charts can be added by clicking on the respective icon in the *Charts* toolbar or the corresponding *Chart*

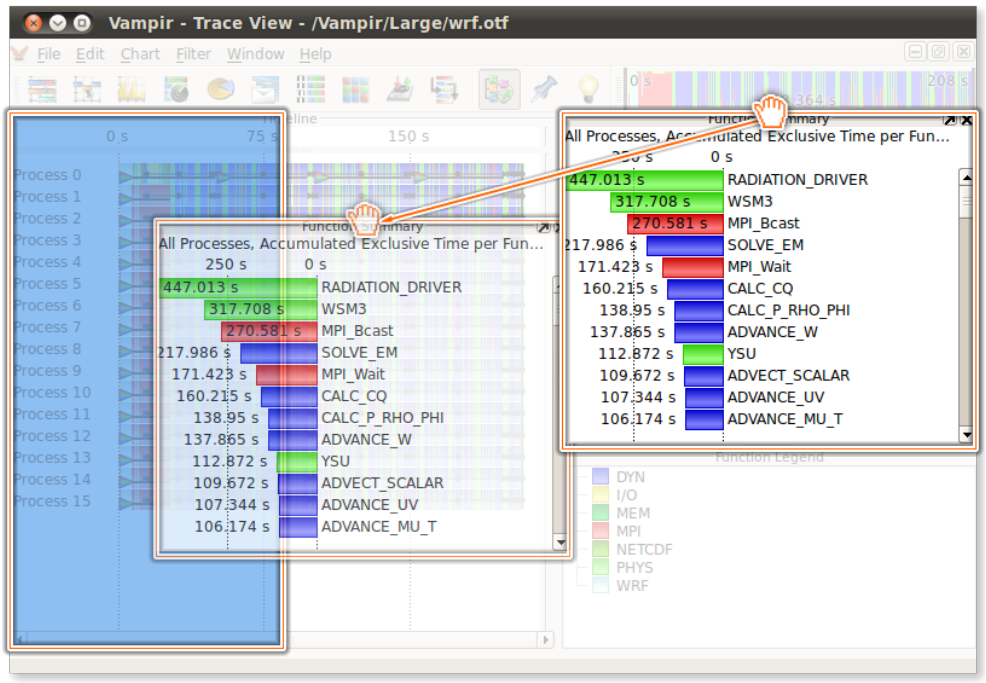


Figure 3.3: Moving and Arranging Charts in the Trace View Window (1)

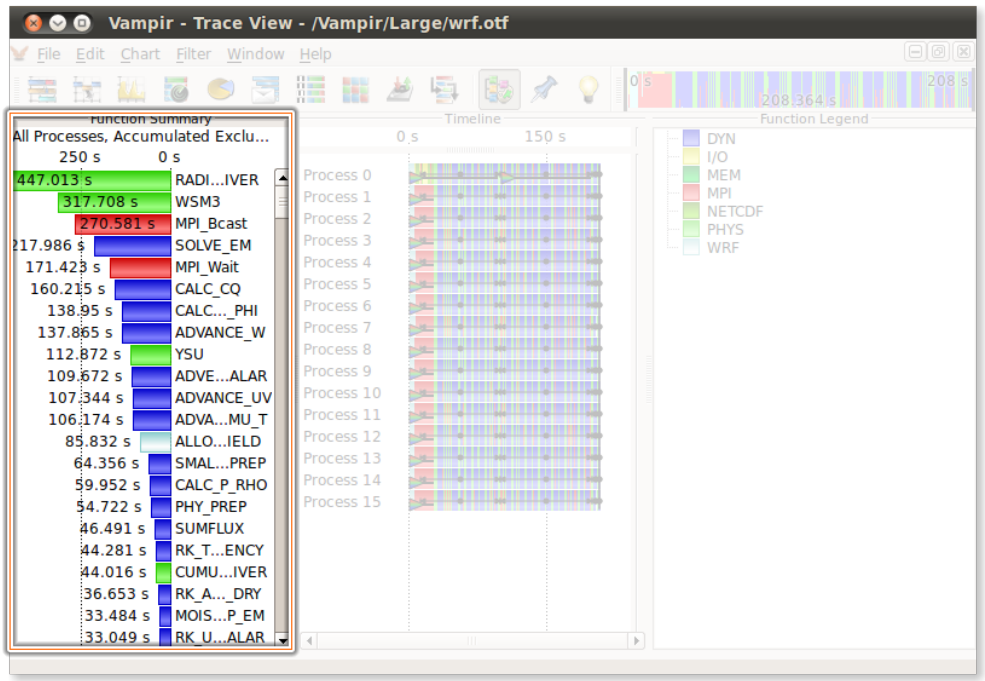


Figure 3.4: Moving and Arranging Charts in the Trace View Window (2)

menu entry. With a few more clicks, charts can be combined to a custom chart arrangement as depicted in Figure 3.2. Customized layouts can be saved as described in Chapter 7.3.

Every chart can be undocked or closed by clicking the dedicated icon in its upper right corner as shown in Figure 3.5. Undocking a chart means to free the chart from the current arrangement and present it in an own window. To dock/undock a chart follow Figure 3.6, respectively Figure 3.7.

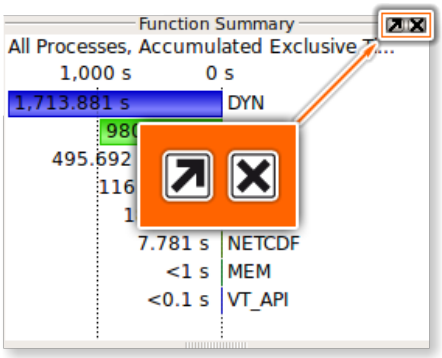


Figure 3.5: Closing (right) and Undocking (left) of a Chart

Considering that labels, e.g., those showing names or values of functions often need more space to show its whole text, there is a further option of resizing. In order to read labels completely, it might be useful to alter the distribution of space shared by the labels and the graphical representation in a chart. When hovering the blank space between labels and graphical representation, a movable separator appears. By dragging the separator decoration with the left mouse button the chart space provided for the labels can be resized. The whole process is illustrated in Figure 3.8.

3.2 Context Menus

All chart displays have their own context menu containing common as well as display specific entries. In this section only the most common entries will be discussed. A context menu can be accessed by right clicking anywhere in the chart window.

Common entries are:

- **Reset Zoom:** Go back to the initial state in horizontal zooming.
- **Reset Vertical Zoom:** Go back to the initial state in vertical zooming.
- **Set Metric:** Set the values which should be represented in the chart, e.g., change from *Exclusive Time* to *Inclusive Time*.

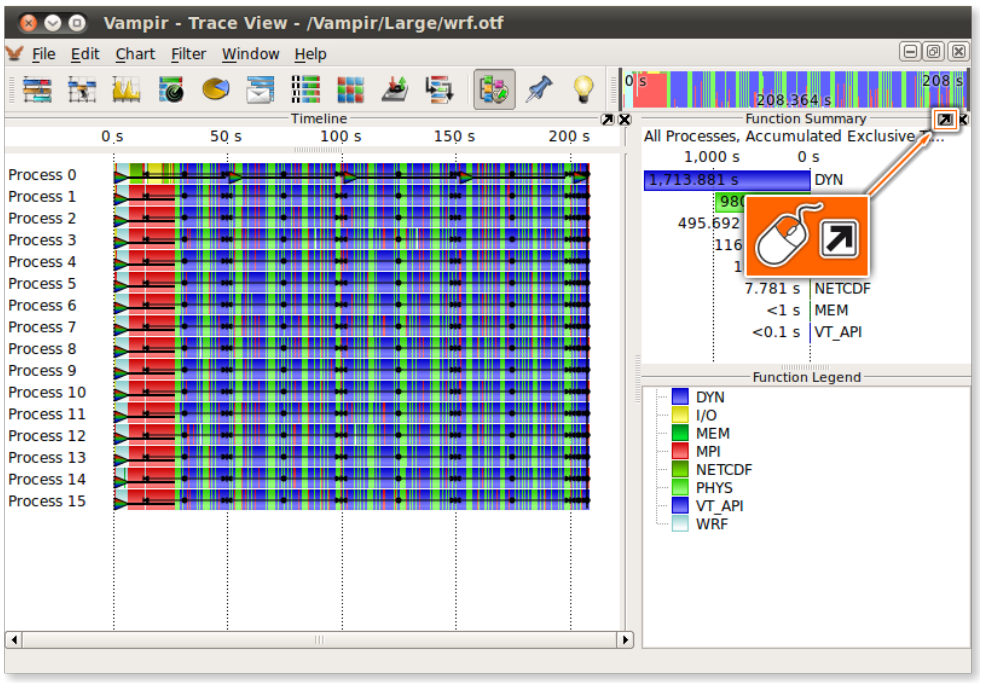


Figure 3.6: Undocking of a Chart

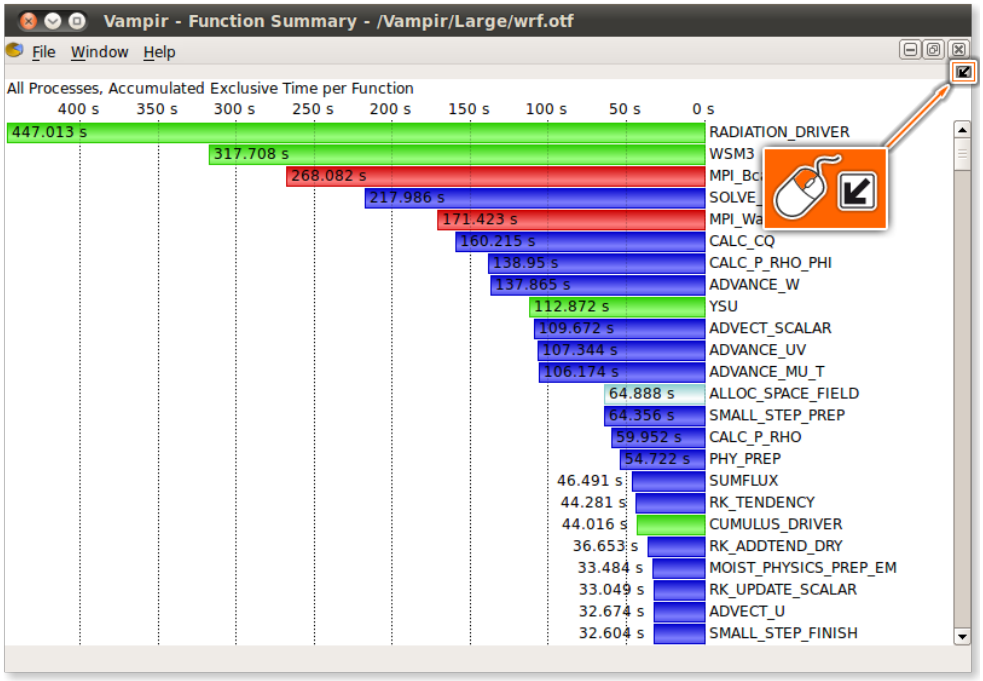


Figure 3.7: Docking of a Chart

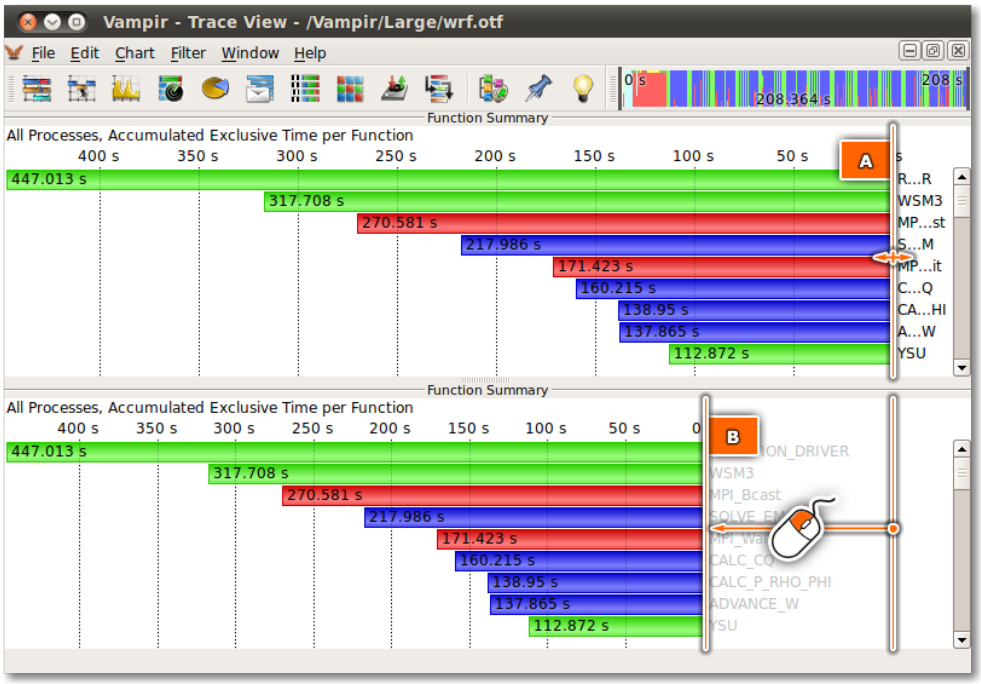


Figure 3.8: Resizing Labels: (A) Hover a Separator Decoration; (B) Drag and Drop the Separator

- **Sort By:** Rearrange values or bars by a certain characteristic.

3.3 Zooming

Zooming is a key feature of Vampir. In most charts it is possible to zoom in and out to get detailed or abstract views of the visualized data.

In the timeline charts zooming produces a more detailed view of a selected time interval and therefore reveals new information that was previously hidden in the larger section. Short function calls in the *Master Timeline* may not be visible unless an appropriate zooming level has been reached. In other words, if the execution time of functions is too short with respect to the available pixel resolution of your computer display, zooming into a shorter time interval is required in order to make them visible.

Note: Other charts are affected by zooming in the timeline displays. The interval chosen in a timeline chart, such as *Master Timeline* or *Process Timeline* also defines the time interval for the calculation of accumulated measurements in the statistical charts.

Statistical charts like the *Function Summary* provide zooming of statistic values. In these cases zooming does not affect any other chart. Zooming is disabled in the *Pie*

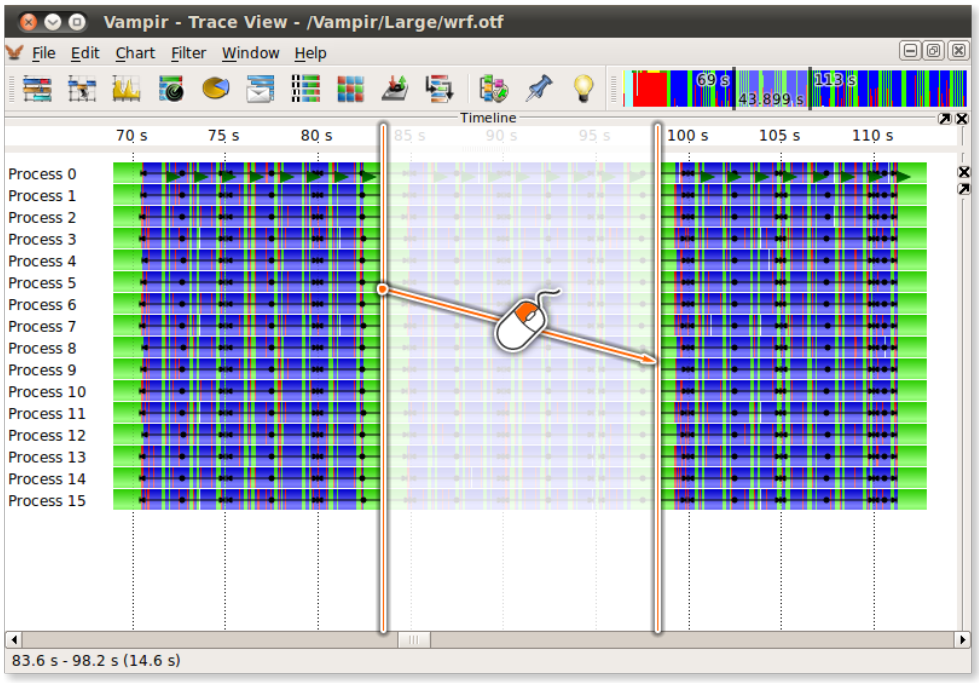


Figure 3.9: Zooming within a Chart

Chart mode of the *Function Summary* accessible via the context menu under *Set Chart Mode* → *Pie Chart*.

To zoom into an area, click and hold the left mouse button and select the area as shown in Figure 3.9. It is possible to zoom horizontally and in some charts also vertically. In the *Master Timeline* horizontal zooming defines the time interval to be visualized whereas vertical zooming selects a group of processes to be displayed. To scroll horizontally move the slider at the bottom or use the mouse wheel. To get back to the initial state of zooming select *Reset Horizontal Zoom* or *Reset Vertical Zoom* (see Section 3.2) in the context menu of the respective performance chart.

Additionally the zoom can be accessed with help of the *Zoom Toolbar* by dragging the borders of the selection rectangle or by scrolling of the mouse wheel as described in Chapter 3.4.

In order to return to the previous zooming state an undo functionality, accessible via the *Edit* menu, is provided. Alternatively, the key combination *Ctrl+Z* also reverts the last zoom. Accordingly, a reverted zooming action can be redone by selecting *Redo* in the *Edit* menu or by pressing *Ctrl+Shift+Z*. The undo functionality is not bound to single performance charts but works across the entire application. The labels of the *Undo* and *Redo* menu entries also state which kind of action will be undone/redone next.

3.4 The Zoom Toolbar

Vampir provides a *Zoom Toolbar* that can be used for zooming and navigation in the trace data. It is located in the upper right corner of the *Trace View* window, shown in Figure 3.1. It is possible to adjust its position via drag and drop. The *Zoom Toolbar* offers an overview and summary of the loaded trace data. The currently zoomed area is highlighted as a rectangle within the *Zoom Toolbar*. By dragging of the two boundaries of the highlighted rectangle the horizontal zooming state can be adjusted.

Note: Instead of dragging boundaries it is also possible to use the mouse wheel for zooming. Hover the *Zoom Toolbar* and scroll up and down to zoom in and out, respectively.

Dragging the zoom area changes the section that is displayed without changing the zoom factor. For dragging, click into the highlighted zoom area and drag and drop it to the desired position. Zooming and dragging within the *Zoom Toolbar* is illustrated in Figure 3.10. If the user double clicks in the *Zoom Toolbar*, the initial zooming state is reverted.

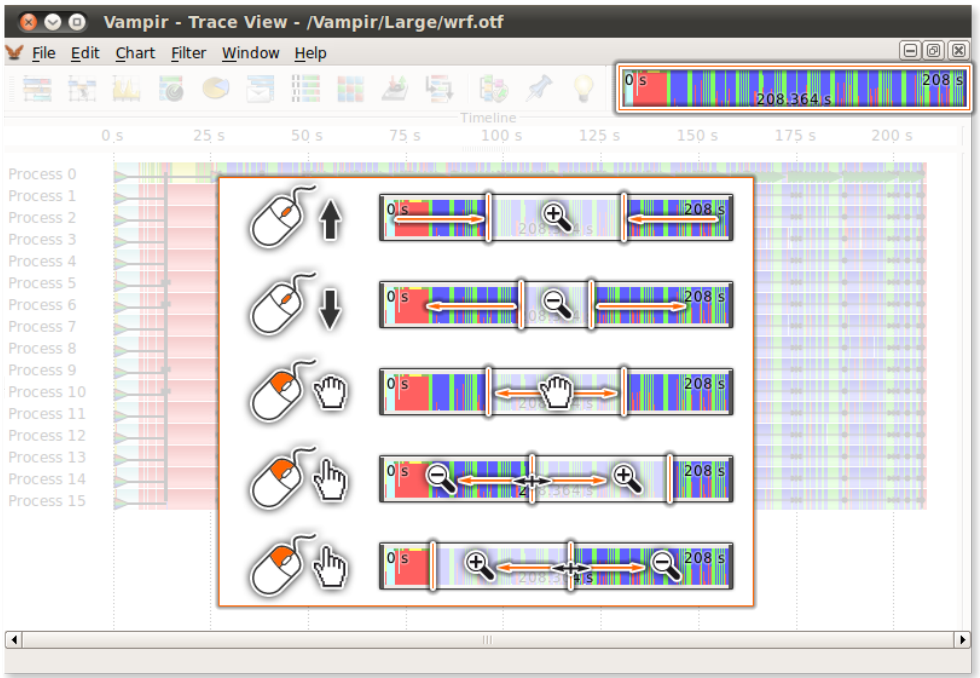


Figure 3.10: Zooming and Navigation within the Zoom Toolbar: (A+B) Zooming in/out with the Mouse Wheel; (C) Scrolling by Moving the Highlighted Zoom Area; (D) Zooming by Selecting and Moving a Boundary of the Highlighted Zoom Area

The colors represent user-defined groups of functions or activities. Please note that all charts added to the *Trace View* window will calculate their statistic information ac-

cording to the selected time interval (zooming state) in the *Zoom Toolbar*. The *Zoom Toolbar* can be enabled and disabled with the toolbar's context menu entry *Zoom Toolbar*.

3.5 The Charts Toolbar

Icon	Name	Description
	Master Timeline	Section 4.1.1
	Process Timeline	Section 4.1.1
	Counter Data Timeline	Section 4.1.2
	Performance Radar	Section 4.1.3
	Function Summary	Section 4.2.1
	Message Summary	Section 4.2.3
	Process Summary	Section 4.2.2
	Communication Matrix View	Section 4.2.4
	I/O Summary	Section 4.2.5
	Call Tree	Section 4.2.6
	Function Legend	Section 4.3.1
	Marker View	Section 4.3.2
	Context View	Section 4.3.3

Table 3.1: Icons of the Charts Toolbar

The *Charts Toolbar* is used to open instances of the available performance charts. It is located in the upper left corner of the *Trace View* window as shown in Figure 3.1. The toolbar can be dragged and dropped to alternative positions. The *Charts Toolbar* can be disabled with the toolbar's context menu entry *Charts*.

Table 3.1 gives an overview of the available performance charts with their corresponding icons. The icons are arranged in three groups, divided by small separators. The first group represents timeline charts, whose zooming states affect all other charts. The second group consists of statistical charts, providing special information and statistics for a chosen interval. Vampir allows multiple instances for charts of these categories. The last group comprises of informational charts, providing specific textual information or legends. Only one instance of an informational chart can be opened at a time.

3.6 Properties of the Trace File

Vampir provides an info dialog containing important characteristics of the opened trace file. This *Trace Properties* are displayed in the Context View dialog, Section 4.3.3, and can be opened via the main menu under *File* $\rightarrow$ *Get Info*. The information originates from the trace file and includes details such as file name, creator, or the OTF version.

4 Performance Data Visualization

This chapter deals with the different charts that can be used to analyze the behavior of a program and the comparison between different function groups, e.g. *MPI* and *Calculation*. Communication performance issues are regarded in this chapter as well. Various charts address the visualization of data transfers between processes. The following sections describe them in detail.

4.1 Timeline Charts

A very common chart type used in event-based performance analysis is the so-called timeline chart. This chart type graphically presents the chain of events of monitored processes or counters on a horizontal time axis. Multiple timeline chart instances can be added to the *Trace View* window via the *Chart* menu or the *Charts* toolbar.

Note: To measure the duration between two events in a timeline chart Vampir provides a tool called *Ruler*. The Ruler is enabled by default during every zoom operation in a timeline chart. In order to use the Ruler for measurement only, i.e. without performing any zoom, hold the *Shift* key pressed while clicking on any point of interest in a timeline chart and moving the mouse while holding the left mouse button pressed. A ruler like pattern appears in the timeline chart which provides the exact time between the start point and the current mouse position.

4.1.1 Master Timeline and Process Timeline

In the *Master Timeline* and the *Process Timeline* detailed information about functions, communication, and synchronization events is shown. Timeline charts are available for individual processes (*Process Timeline*) as well as for a collection of processes (*Master Timeline*). The *Master Timeline* consists of a collection of rows. Each row represents a single process, as shown in Figure 4.1. A *Process Timeline* shows the different levels of function calls in a stacked bar chart for a single process as depicted in Figure 4.2.

Every timeline row consists of a process name on the left and a colored sequence of function calls or program phases on the right. The color of a function is defined by its group membership, e.g., `MPI_Send()` belonging to the function group *MPI* has the same color, presumably red, as `MPI_Recv()`, which also belongs to the function

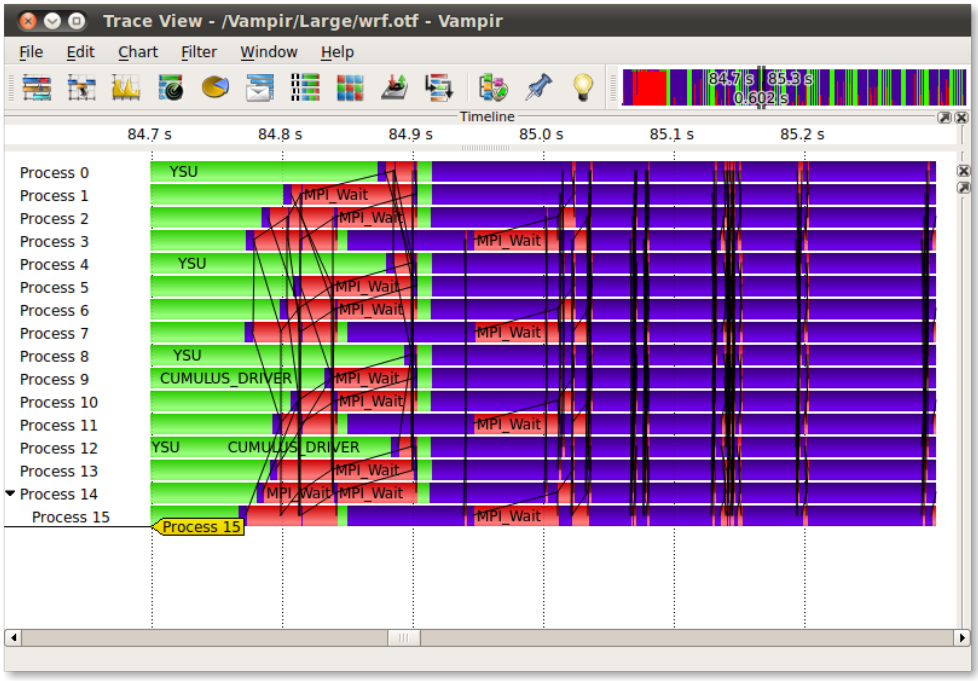


Figure 4.1: Master Timeline

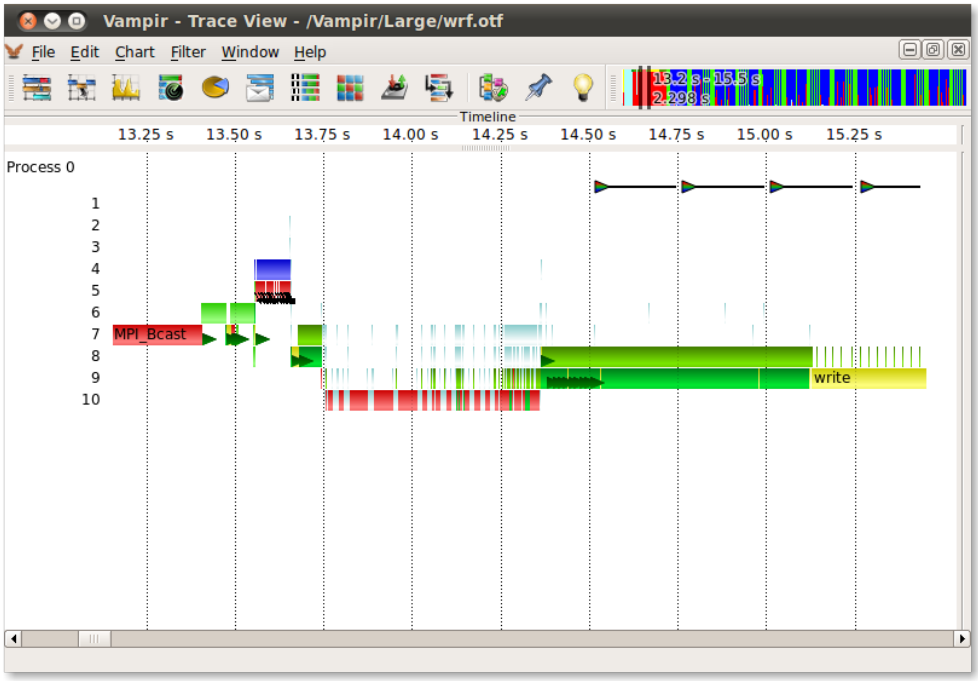


Figure 4.2: Process Timeline

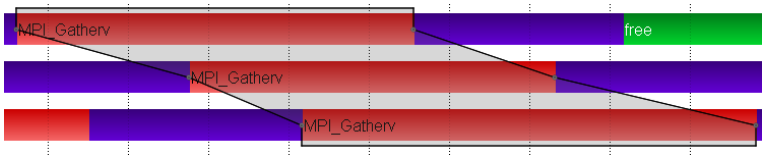


Figure 4.3: Selected MPI Collective in Master Timeline

group *MPI*. Clicking on a function highlights it and causes the *Context View* display to show detailed information about that particular function, e.g., its corresponding function group name, time interval, and the complete name. The *Context View* display is explained in Chapter 4.3.3. Clicking on a process label on the left hand side provides basic information about the process in the *Context View* and highlights the label. Process rows can be re-ordered by clicking and dragging the process label at the front of each row. If a process has been recorded with subordinated information like threads, this information can be hidden and exposed by clicking the black arrow shape in front of the process label.

Some function invocations are very short. Hence, these are not shown in the overall view due to a lack of display pixels. A zooming mechanism is provided to inspect a specific time interval in more detail. For further information on zooming see Section 3.3. If zooming has been performed, scrolling in horizontal direction is possible with the mouse wheel or the scroll bar at the bottom.

The *Process Timeline* resembles the *Master Timeline* with slight differences. The chart's timeline is divided into levels, which represent the different call stack levels of function calls. The initial function begins at the first level, a sub-function called by that function is located a level beneath and so forth. If a sub-function returns to its caller, the graphical representation also returns to the level above.

In addition to the display of categorized function invocations, Vampir's *Master-* and *Process Timeline* also provide information about communication events. Messages exchanged between two different processes are depicted as black lines. In timeline charts, the progress in time is reproduced from left to right. The leftmost (starting) point of a message line and its underlying process bar therefore identify the sender of the message, whereas the rightmost position of the same line represents the receiver of the message. The corresponding function calls usually reflect a pair of MPI communication directives like `MPI_Send()` and `MPI_Recv()`. Collective communication like `MPI_Gatherv()` is also displayed in the *Master Timeline* as shown in Figure 4.3.

Furthermore, additional information like message bursts, markers and I/O events is available. Table 4.1 shows the symbols and descriptions of these objects.

Since the *Process Timeline* reveals information of one process only, short black arrows are used to indicate outgoing communication. Clicking on message lines or arrows shows message details like sender process, receiver process, message length, message duration, and message tag in the *Context View* display.

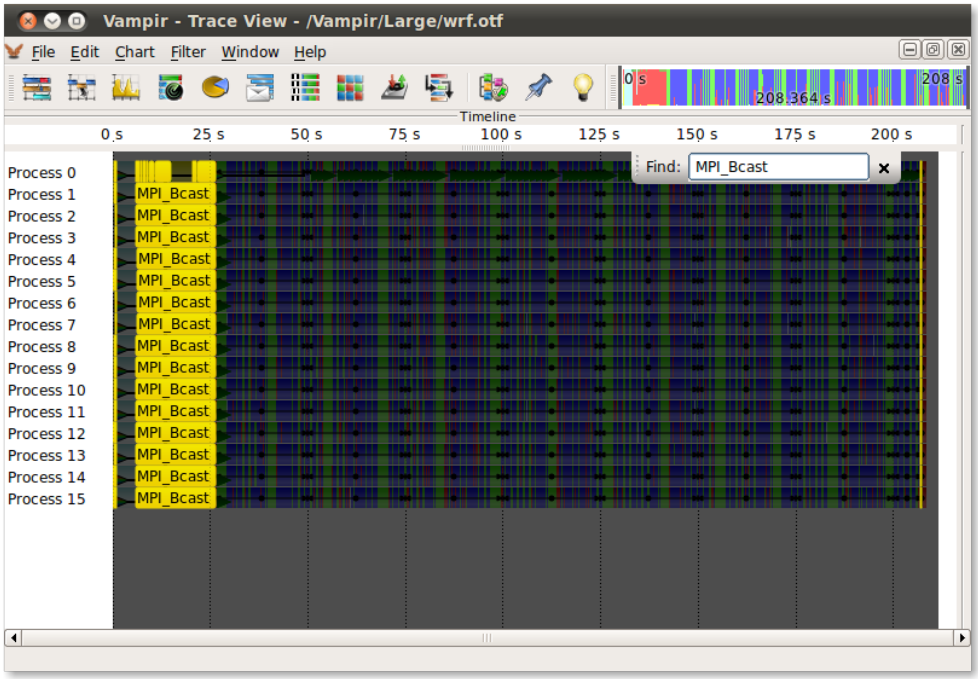


Figure 4.4: Search for MPI_Bcast in the Master Timeline

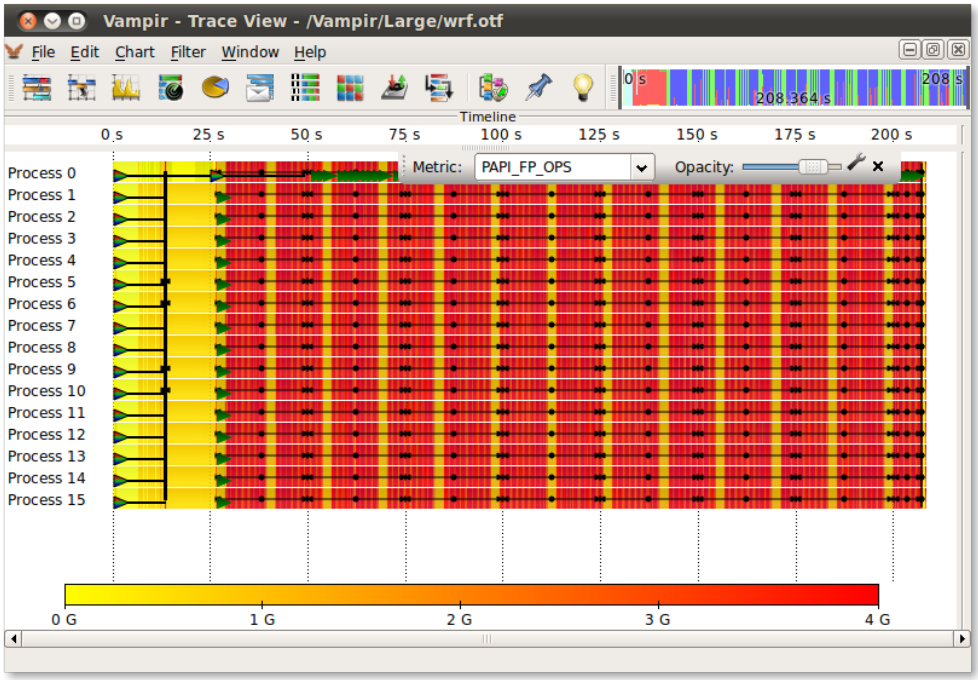


Figure 4.5: Active overlay showing PAPI_FP_OPS in the Master Timeline

Symbol	Description
Message Burst	Due to a lack of pixels it is not possible to display a large amount of messages in a very short time interval. Therefore outgoing messages are summarized as so-called message bursts. In this representation you cannot determine which processes receive these messages. Zooming into this interval reveals the corresponding single messages.
Markers multiple single	To indicate particular points of interest during the runtime of an application, like errors or warnings, markers can be placed in a trace file. They are drawn as triangles which are colored according to their types. To indicate that two or more markers are located at the same pixel, a tricolored triangle is drawn.
I/O Events	Vampir shows detailed information about I/O operations, if they are included in the trace file. I/O events are depicted as triangles at the beginning of an I/O interval. In order to see the whole interval of a single I/O event its triangle has to be selected. In that case a second triangle indicating the end of the interval appears. Multiple I/O events are tricolored and drawn as a triangle with a line to the end of the interval.

Table 4.1: Additional Information in the Master and Process Timeline

The *Master Timeline* also provides the possibility to search for function and function group occurrences. In order to activate the search mode use the context menu and select *Find...*. After activation an input field appears at the top of the *Master Timeline*. A search string can be written in this field and all corresponding function and function group occurrences are highlighted in yellow in the *Master Timeline*. An example search for the function `MPI_Bcast` is depicted in Figure 4.4.

Furthermore, the *Master Timeline* also features an overlay mode for performance counter data, Figure 4.5. In order to activate the overlay mode use the context menu *Options* → *Performance Data*. When the overlay mode is active a control window appears at the top of *Master Timeline*. It allows to select the displayed counter data (metric). The counter data is displayed in a color coded fashion like in the *Performance Radar*, Section 4.1.3. The color scale can be freely customized by clicking on the wrench icon. The control window also provides an opacity control slider. This slider allows to adjust the opacity of the overlay and thus makes the underlying functions easily visible without the need to disable the overlay mode.

4.1.2 Counter Data Timeline

Counters are values collected over time to count certain events like floating point operations or cache misses. Counter values can be used to store not just hardware performance counters but arbitrary sample values. There can be counters for different statistical information as well, for instance counting the number of function calls or a value in an iterative approximation of the final result. Counters are defined during the instrumentation of the application and can be individually assigned to processes.

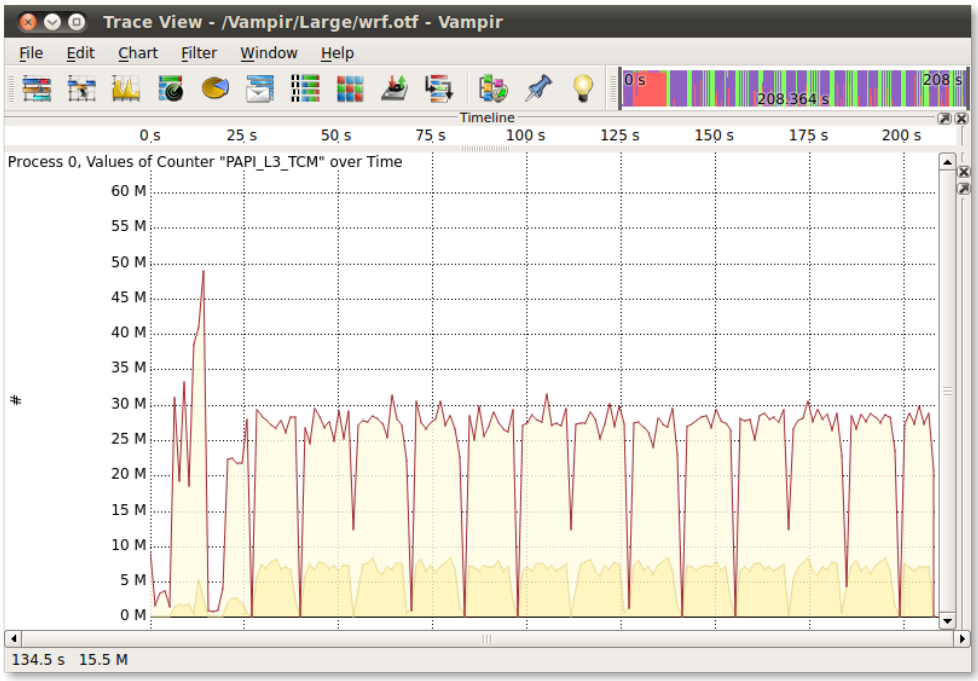


Figure 4.6: Counter Data Timeline

An example *Counter Data Timeline* chart is shown in Figure 4.6. The chart is restricted to one counter at a time. It shows the selected counter for one measuring point (e.g., process). Using multiple instances of the *Counter Data Timeline*, counters or processes can be compared easily.

The displayed graph in the chart is constructed from actual measurements (data points). Since display space is limited it is likely that there are more data points than display pixels available. In that case multiple data points need to be displayed on one pixel (width). Therefore the counter values are displayed in two graphs. A maximum line (red) and an average line (yellow). When multiple data points need to be displayed on one pixel width, the red line shows the data point with the highest value, and the yellow line indicates the average of all data points lying on this pixel width. An optional blue line shows the lowest value. When zooming into a smaller time range less data points need to be displayed on the available pixel space. Eventually, when zooming

far enough only one data point needs to be display on one pixel. Then also the three graphs will merge together. The actual measured data points can be displayed in the chart by enabling them via the context menu under *Options...*.

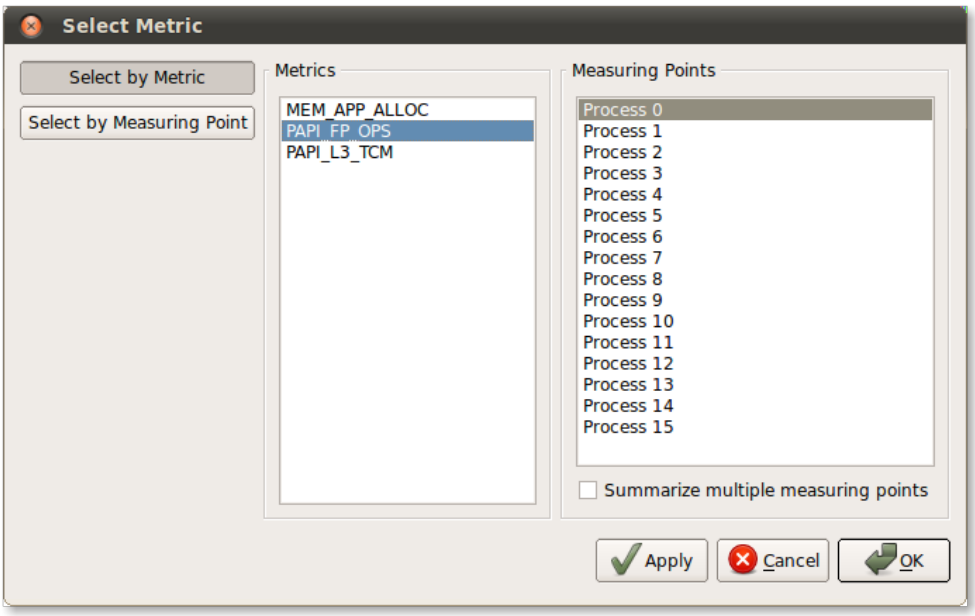


Figure 4.7: Select metric dialog

The context menu entry *Select Metric...* opens the selection dialog depicted in Figure 4.7. This dialog allows to choose the displayed counter in the chart. Each counter is defined by its metric and its measuring point. Note, depending on the measurement not all metrics might be available on all measurement points.
The two left buttons in the dialog decide whether the counter should be selected by metric or by measuring point first. In the case of *Select by Metric* there is also the option to *Summarize multiple measuring points* available. This option allows to identify outliers by summarizing counters (e.g., PAPI_FP_OPS) over multiple measuring points (e.g., processes). Hence, when this option is active multiple measuring points can be selected. The counter for the selected metric is then summarized over all selected measuring points. The displayed counter graphs in the chart need then to be read as follows. The yellow *average* line in the middle displays the average value (e.g., *PAPI_FP_OPS*) of all selected measuring points (e.g., processes) at a given time. The red *maximum* line shows the highest value that one of the selected measuring points achieved at a given time. A click with the left mouse button on any point in the chart reveals its details in the *Context View* display. Stated are the minimum, maximum, and average values and the measurement points (e.g., processes) that achieved maximum and minimum values at the selected point in time.

The options dialog is depicted in Figure 4.8. It is accessible via the context menu under *Options...*. It allows to enable and disable the display of the graph's line, data points,

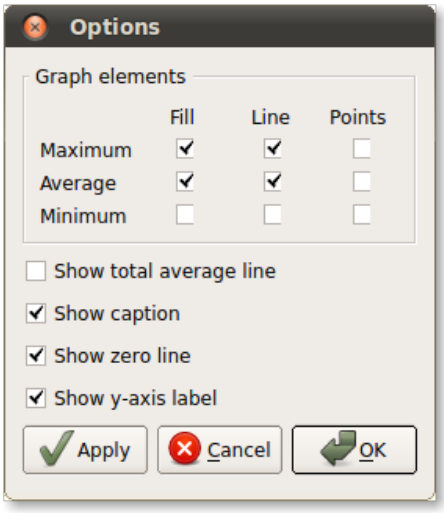


Figure 4.8: Counter Timeline options dialog

and filling. It is also possible to enable an average line showing the average value of all data points in the visible area. Likewise, the chart's caption and y-axis label can turned on and off. The switch *Show zero line* disables the auto-scaling of the y-axis for the lower bound and enforces a zero line in all situations.

The *Counter Data Timeline* chart allows to create custom metrics. This process is described in Section 4.1.3. Created custom metrics become available in the *Select Metric* dialog.

4.1.3 Performance Radar

The *Performance Radar* chart, Figure 4.9, displays counter data and provides the possibility to create custom metrics. In contrast to the Counter Data Timeline the Performance Radar shows one counter for all processes at once. The values of the counter are displayed in a color-coded fashion.

The displayed counter in the chart can be chosen via the context menu entry *Set Metric*. Own created custom metrics are listed under this option as well.

The option *Adjust Bar Height to* allows to change the height of the displayed value bars in the chart. This useful for traces with a large number of processes. Here the option *Adjust Bar Height to* → *Fit Chart Height* tries to display all processes in the chart. This provides an overview of the counter data across the entire application run.

Set Chart Mode allows to define whether minimum, maximum, or average values should be shown. This setting comes into effect when multiple measured data points need to be displayed on one pixel. If *Maximum* or *Minimum* is active, the data point

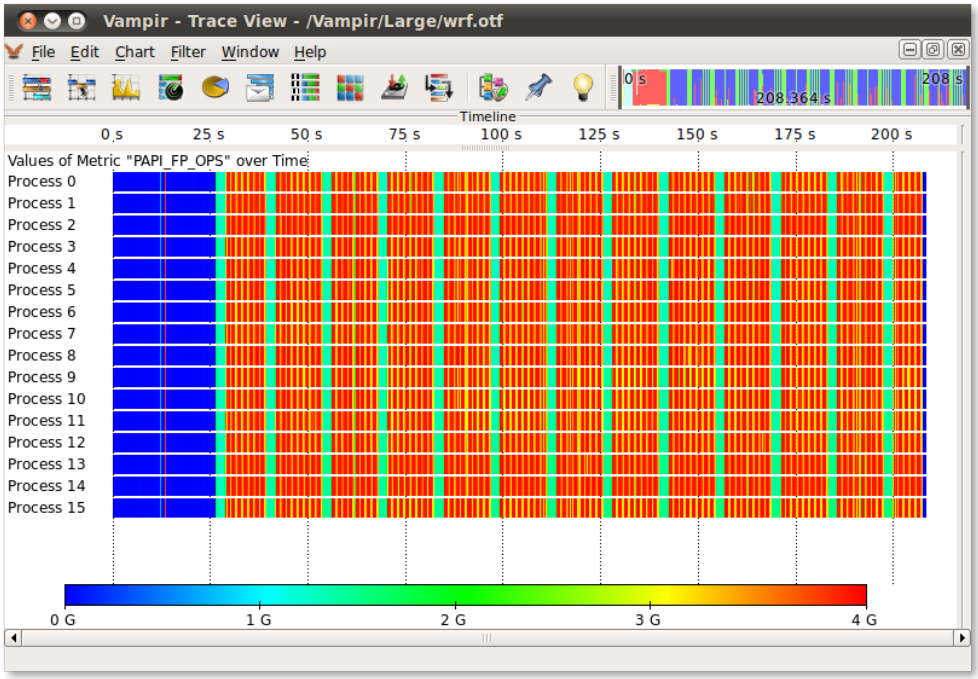


Figure 4.9: Performance Radar

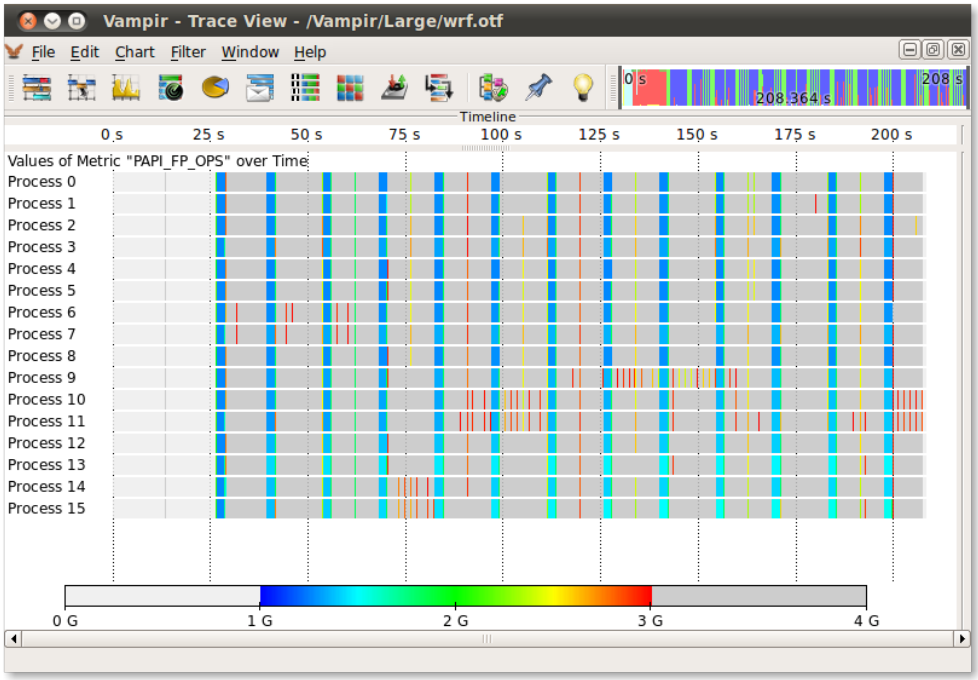


Figure 4.10: Adjusted value range in color scale

with the highest or lowest value is displayed, respectively. In case of *Average* the average of all data points on the respective pixel width is displayed. This procedure is also explained in section *Counter Data Timeline* 4.1.2.

The value range of the color scale can be easily adjusted with the left mouse button. To adjust the color-coded value rage just drag the edges of the color scale to the desired positions. Figure 4.10 depicts the Performance Radar chart shown in Figure 4.9 with a smaller value range of 1 G - 3 G FLOPS. This allows to easily spot areas of high or low performance in the trace file. The selected value range can also be dragged to other positions in the color scale. A double-click with the left mouse button on the color scale resets the selected value range.
The option *Options* → *Color Scale...* in the context menu of the chart allows to customize the color scale to the own preferences.

Master Timeline Overlay Mode

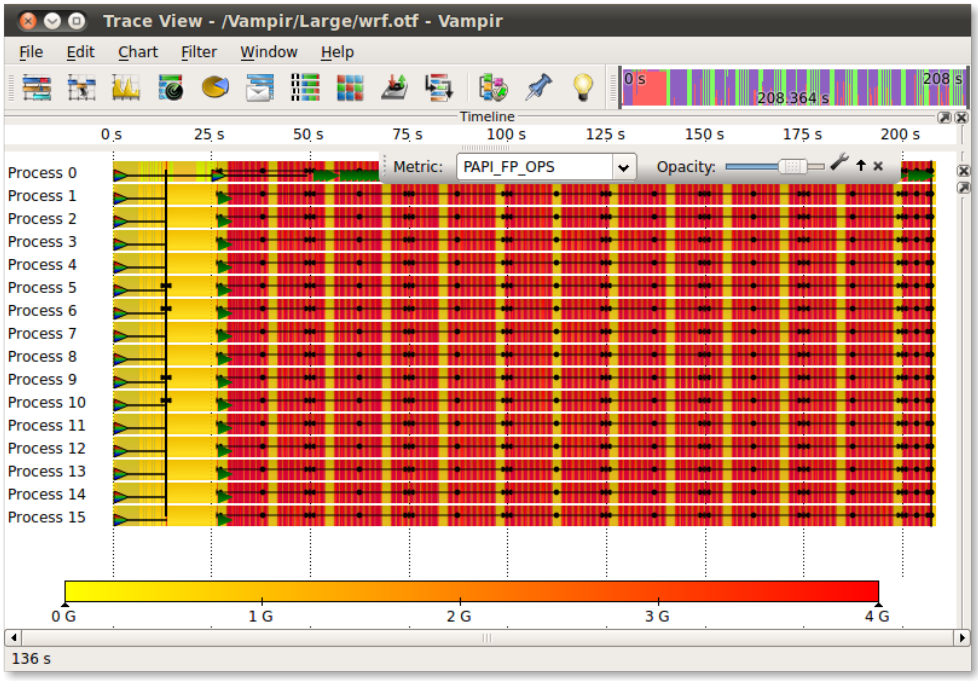


Figure 4.11: Master Timeline with active performance data overlay

Figure 4.11 shows an overview of the performance data overlay mode available in the Master Timeline chart. The overlay is capable of displaying all metrics available in the Performance Radar chart and the Counter Data Timeline chart. It is activated via the chart's context menu under *Options* → *Performance Data*. When the overlay mode is active, a control window appears at the top of Master Timeline chart. It allows to configure the overlay and to select the displayed performance data (metric).

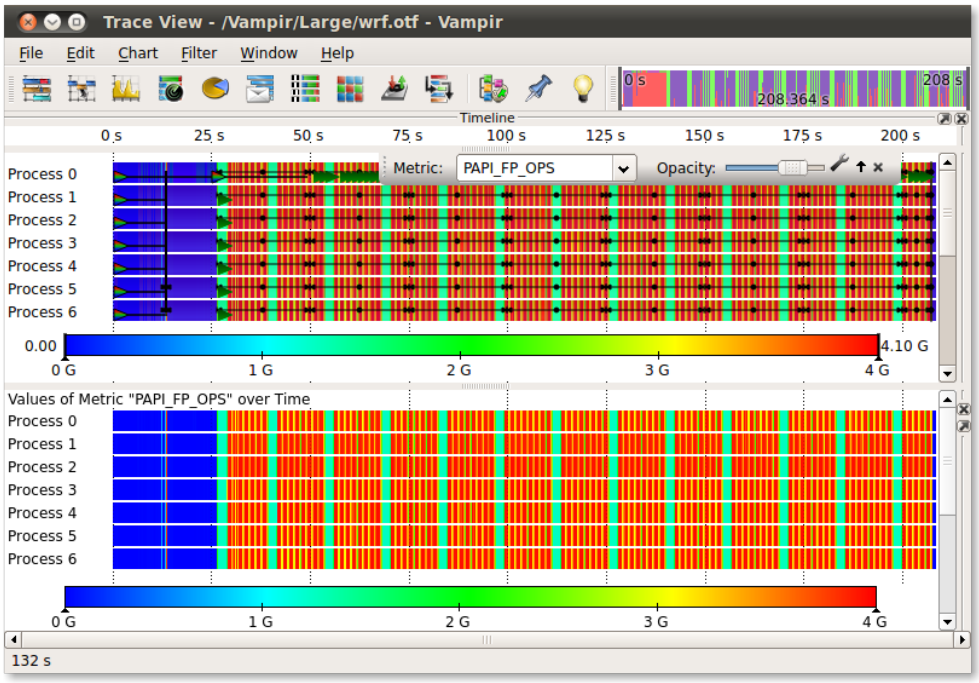


Figure 4.12: Master Timeline (top chart) and Performance Radar (bottom chart) displaying the same PAPI_FP_OPS counter

The selected metric is shown in a color-coded fashion like in the Performance Radar chart. Figure 4.12 depicts the Master Timeline chart (top) and the Performance Radar chart (bottom), both displaying the same performance metric PAPI_FP_OPS (floating point operations per second). As can be seen, the overlay mode provides the performance data visualization capabilities of the Performance Radar for the Master Timeline. To fully benefit from this combination the opacity slider of the overlay control window should be used, see Figure 4.13. The slider allows to quickly manipulate the opacity of the overlay and thus making underlying functions visible. This is particularly useful for first pinpointing performance relevant areas and then directly analyzing the individual identified functions in the Master Timeline.

The color scale of the performance data overlay is freely customizable. Clicking the wrench icon in the overlay control window opens the color scale options dialog. The color scale provides three modes: *Default*, *Highlight*, and *Find*. Additionally, the *Custom* mode allows to manually adapt the color scale to the own preferences.

Examples

This section illustrates the usage of the Performance Radar chart and the Master Timeline overlay in a few examples. The trace file used for the examples shows a WRF

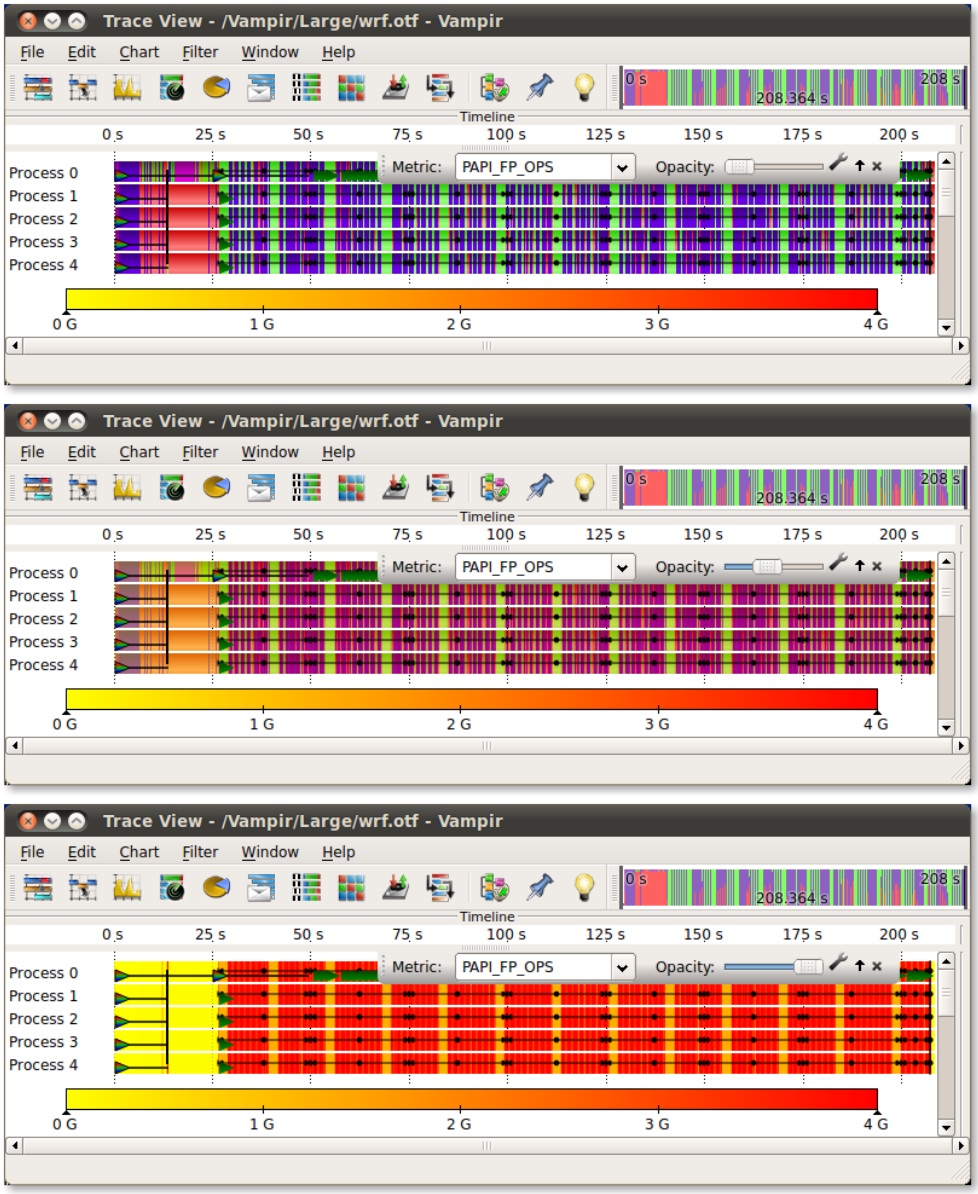

Trace View - /Vampir/Large/wrf.otf - Vampir
File Edit Chart Filter Window Help
0 s
208.364 s
208 s
Timeline
0 s 25 s 50 s 75 s 100 s 125 s 150 s 175 s 200 s
Metric: PAPI_FP_OPS
Opacity:
Process 0
Process 1
Process 2
Process 3
Process 4
0 G 1 G 2 G 3 G 4 G
Trace View - /Vampir/Large/wrf.otf - Vampir
File Edit Chart Filter Window Help
0 s
208.364 s
208 s
Timeline
0 s 25 s 50 s 75 s 100 s 125 s 150 s 175 s 200 s
Metric: PAPI_FP_OPS
Opacity:
Process 0
Process 1
Process 2
Process 3
Process 4
0 G 1 G 2 G 3 G 4 G
Trace View - /Vampir/Large/wrf.otf - Vampir
File Edit Chart Filter Window Help
0 s
208.364 s
208 s
Timeline
0 s 25 s 50 s 75 s 100 s 125 s 150 s 175 s 200 s
Metric: PAPI_FP_OPS
Opacity:
Process 0
Process 1
Process 2
Process 3
Process 4
0 G 1 G 2 G 3 G 4 G

Figure 4.13: Image series showing different opacity settings for the performance data overlay, going from zero opacity in the top image to full opacity in the bottom image

weather forecast code run. The timelines show the initialization in the beginning followed by a number of compute iterations. Figure 4.13 depicts this trace file. The top image shows the pure timelines of the Master Timeline chart, the bottom image shows the values of the PAPI_FP_OPS counter superimposed on the timelines. Here, the red areas indicate high computational activity and therefore mark the compute iterations.

High and Low FLOP Rate

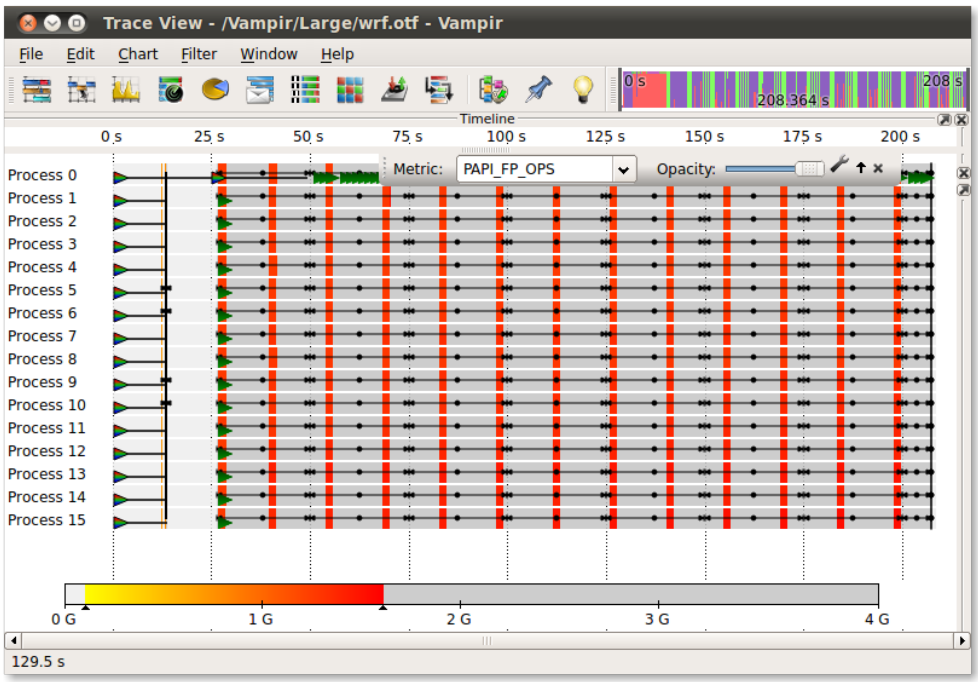


Figure 4.14: Highlighted areas with a low FLOP rate

In order to analyze the FLOP rate, the overlay mode of the Master Timeline is configured to show the performance counter PAPI_FP_OPS. To identify functions with a high or low FLOP rate the value range of the color scale can be limited. This is done by dragging the edges of the colored area of the scale to the desired minimum/maximum values. That way only values inside the chosen range appear color-coded in the chart. Outside values are visualized in gray.

Figure 4.14 and Figure 4.15 depict two examples. Functions with a low FLOP rate are highlighted in Figure 4.14. The color scale is limited to a range between 100 M and 1.6 G FLOPS. The minimum value is raised to 100 M in order to gray out non-computing functions like MPI. In Figure 4.14 all areas with a low FLOP rate are highlighted in red. In this example these areas represent functions in the beginning of each iteration. Functions with a high FLOP rate are highlighted in Figure 4.15. Here the color scale

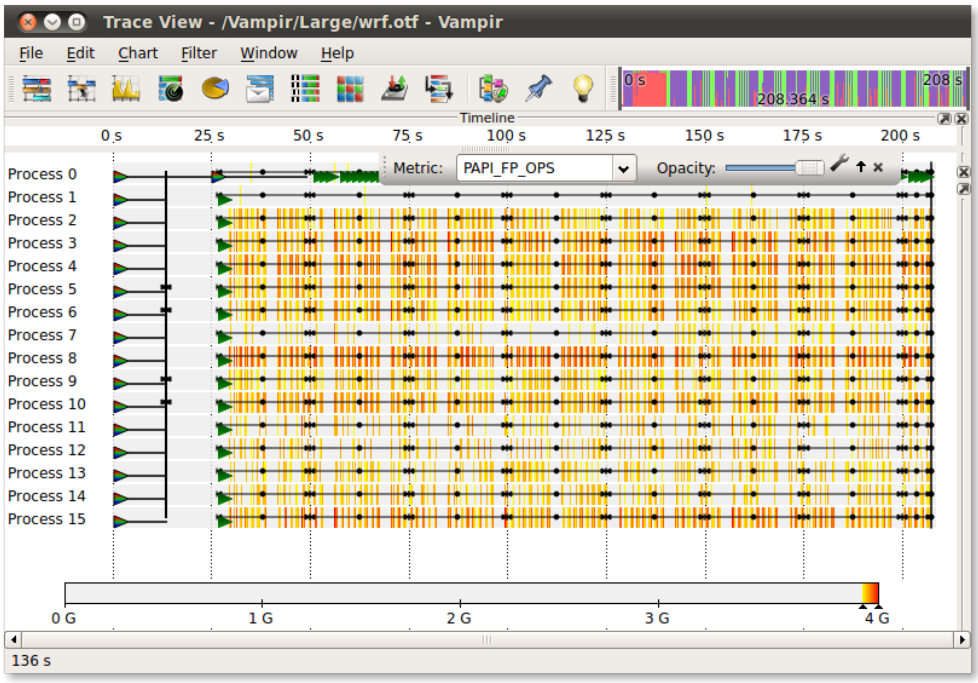


Figure 4.15: Highlighted areas with a high FLOP rate

is set to highlight only areas with the highest FLOP rate. These areas are represented by functions in the compute iterations.

Memory Allocation

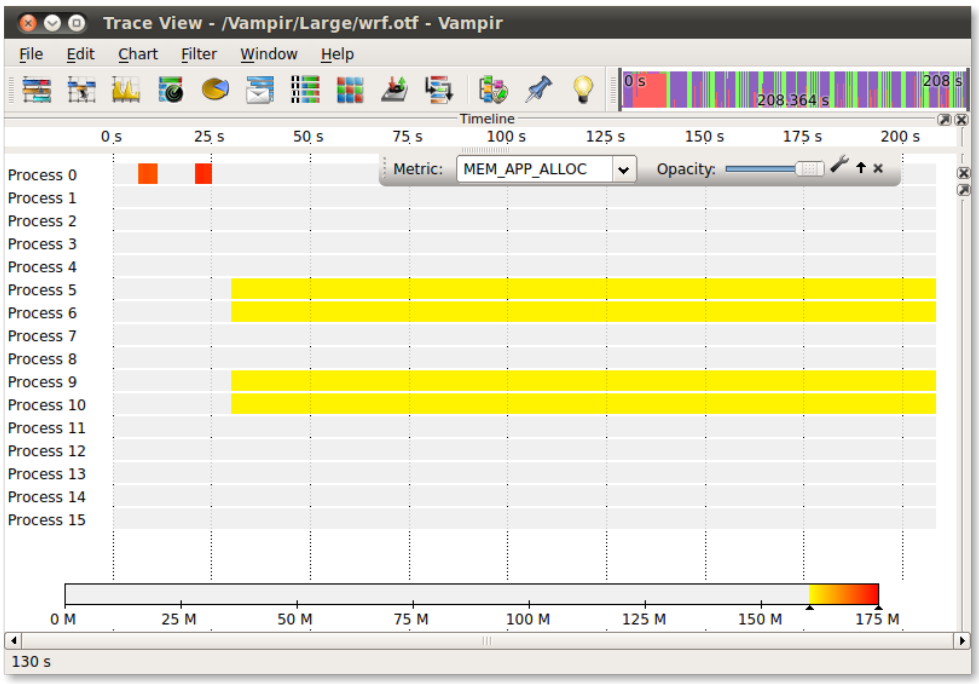


Figure 4.16: Functions with 160 MB - 175 MB allocated memory

The performance data overlay can also be used to identify functions with a certain amount of allocated memory. Figure 4.16 shows an example. Here functions that have between 160 MB and 175 MB memory allocated are highlighted. The highlighted range of allocated memory can be easily changed by adjusting the color scale value range.

4.2 Statistical Charts

4.2.1 Function Summary

The *Function Summary* chart, Figure 4.17, gives an overview of the accumulated time consumption across all function groups and functions. For example every time a process calls the `MPI_Send()` function the elapsed time of that function is added to the *MPI* function group time. The chart gives a condensed view of the execution of the application. A comparison between the different function groups can be made and dominant function groups can be distinguished easily.

It is possible to change the information displayed via the context menu entry *Set Metric* that offers options like *Average Exclusive Time*, *Number of Invocations*, *Accumulated Inclusive Time*, etc.

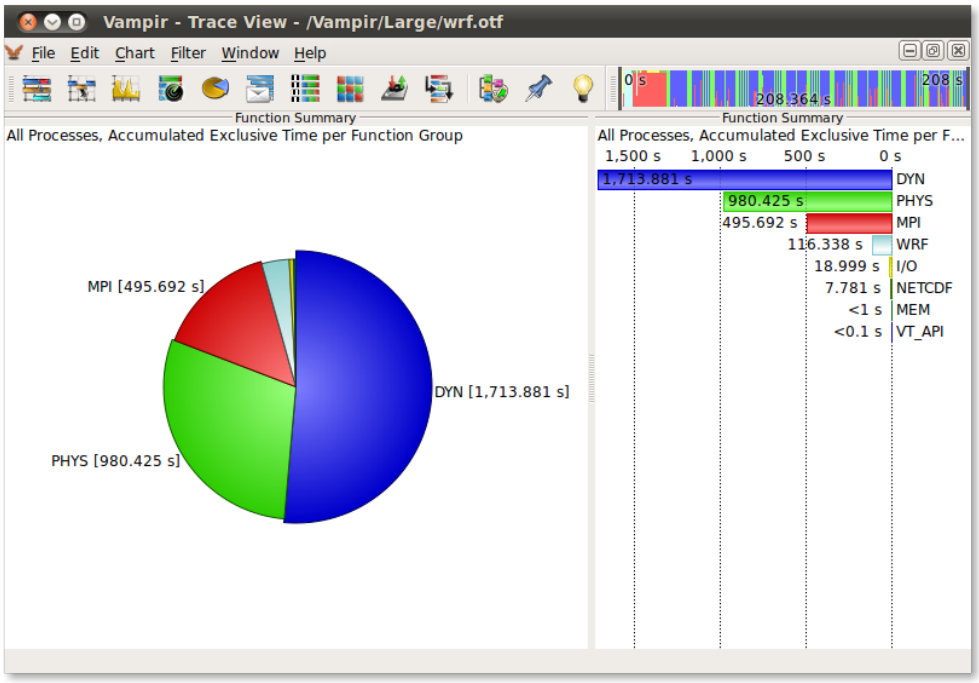


Figure 4.17: Function Summary

Note: *Inclusive* means the amount of time spent in a function and all of its subroutines. *Exclusive* means the amount of time spent in just this function.

The context menu entry *Set Event Category* specifies whether function groups or functions should be displayed in the chart. The functions own the color of their function group.

It is possible to hide functions and function groups from the displayed information with the context menu entry *Filter*. In order to mark the function or function group to be filtered just click on the associated label or color representation in the chart. Using the *Process Filter* (see Section 5.1) allows you to restrict this chart to a set of processes. As a result, only the consumed time of these processes is displayed for each function group or function. Instead of using the filter which affects all other displays by hiding processes, it is possible to select a single process via *Set Process* in the context menu of the *Function Summary*. This does not have any effect on other charts.

The *Function Summary* can be shown as *Histogram* (a bar chart, like in timeline charts) or as *Pie Chart*. To switch between these representations use the *Set Chart Mode* entry of the context menu.

The shown functions or function groups can be sorted by name or value via the context menu option *Sort By*.

4.2.2 Process Summary

The *Process Summary*, depicted in Figure 4.18, is similar to the *Function Summary* but shows the information for every process independently. This is useful for analyzing the balance between processes to reveal bottlenecks. For instance finding that one process spends a significantly high time performing the calculations could indicate an unbalanced distribution of work and therefore can slow down the whole application.

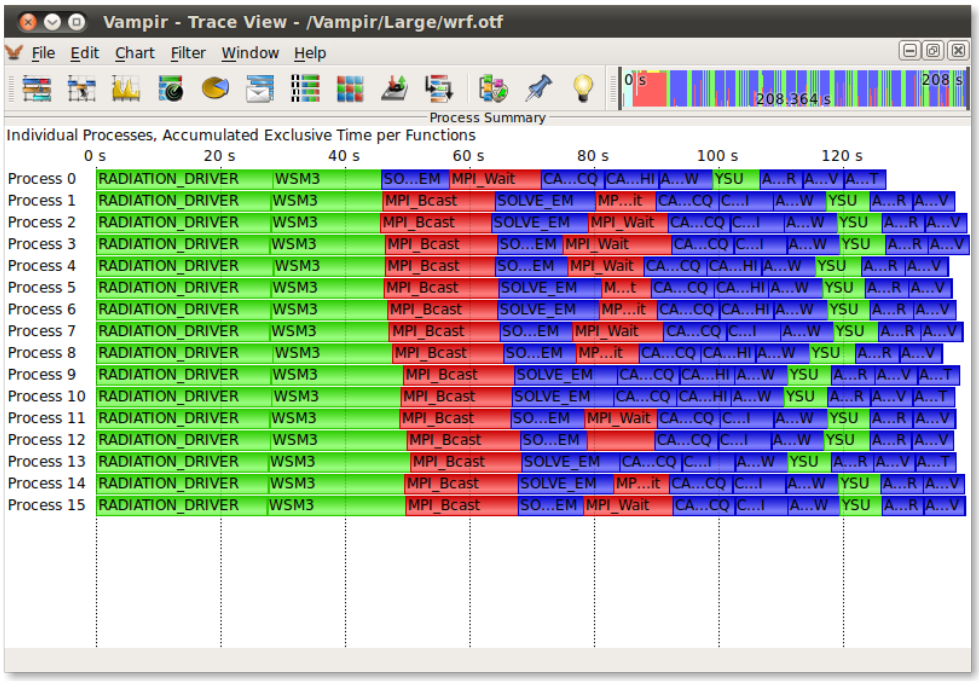


Figure 4.18: Process Summary

The context menu entry *Set Event Category* specifies whether function groups or functions should be displayed in the chart. The functions own the color of their function group.

The chart calculates statistics based on *Number of Invocations*, *Accumulated Inclusive Time*, or *Accumulated Exclusive Time*. To change between these three modes use the context menu entry *Set Metric*.

The number of clustered profile bars is based on the chart height by default. You can also disable the clustering or set a fixed number of clusters via the context menu entry *Clustering* by selecting the corresponding value in the spin box. Located left of the clustered profile bars is a graphical overview indicating the processes associated to the cluster. Moving the cursor over the blue areas in the overview opens a tooltip stating the respective process name.

It is possible to profile only one function or function group or to hide arbitrary functions and function groups from the displayed information. To mark the function or function

group to be profiled or filtered just click on the associated color representation in the chart. The context menu entries *Profile of Selected Function/(Group)* and *Filter Selected Function/(Group)* will then provide the possibility to profile or filter the selected function or function group. Using the *Process Filter* (see Section 5.1) allows you to restrict this view to a set of processes.

The context menu entry *Sort by* allows you to order function profiles by *Number of Clusters*. This option is only available if the chart is currently showing clusters. Otherwise function profiles are sorted automatically by process. While profiling one function the menu entry *Sort by Value* allows to order functions by their execution time.

4.2.3 Message Summary

The *Message Summary* is a statistical chart showing an overview of all messages grouped by certain characteristics, Figure 4.19.

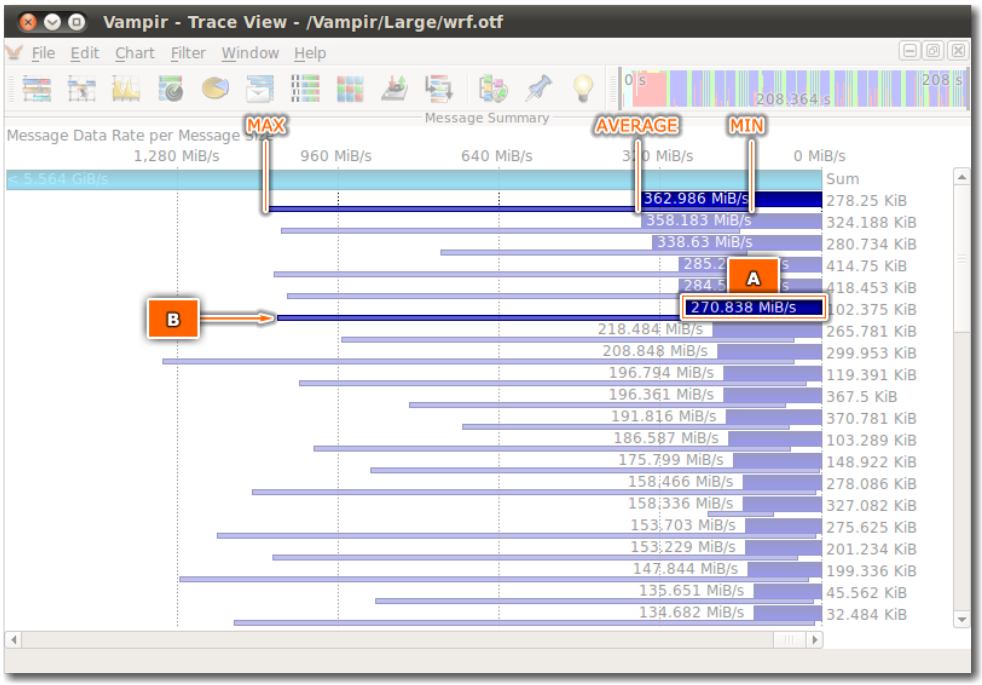


Figure 4.19: Message Summary Chart with metric set to *Message Transfer Rate* showing the average transfer rate (A), and the minimal/maximal transfer rate (B)

All values are represented in a bar chart fashion. The number next to each bar is the group base while the number inside a bar depicts the values depending on the chosen metric. Therefore, the *Set Metric* sub-menu of the context menu can be used to switch between *Aggregated Message Volume*, *Message Size*, *Number of Messages*, and *Message Transfer Rate*.

The group base can be selected via the context menu entry *Group By*. Possible options are *Message Size*, *Message Tag*, and *Communicator (MPI)*.

Note: There will be one bar for every occurring group. However, if the metric is set to *Message Transfer Rate*, the minimal and the maximal transfer rate is given in an additional small bar beneath the main bar showing the average transfer rate. The additional bar starts at the minimal rate and ends at the maximal rate, see Figure 4.19.

In order to filter out messages click on the associated label or color representation in the chart and then choose *Filter* from the context menu.

4.2.4 Communication Matrix View

The *Communication Matrix View* is another way of analyzing communication imbalances. It shows information about messages sent between processes.

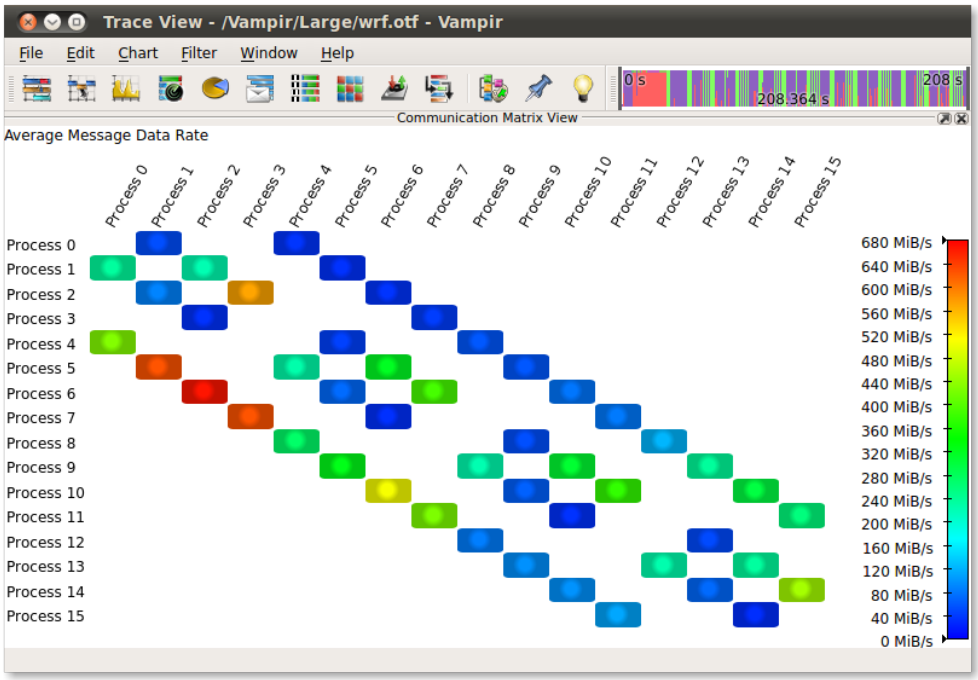


Figure 4.20: Communication Matrix View

The chart, as shown in Figure 4.20, is figured as a table. Its rows represent the sending processes whereas the columns represent the receivers. The color legend on the right indicates the displayed values. It adapts automatically to the currently shown value range.

It is possible to change the type of displayed values. Different metrics like the average duration of messages passed from sender to recipient or minimum and maximum

bandwidth are offered. To change the type of value that is displayed use the context menu option *Set Metric*.

Use the *Process Filter* to define which processes/groups should be displayed. (see Section 5.1).

Note: A high duration is not automatically caused by a slow communication path between two processes, but can also be due to the fact that the time between starting transmission and successful reception of the message can be increased by a recipient that delays reception for some reason. This will cause the duration to increase (by this delay) and the message rate, which is the size of the message divided by the duration, to decrease accordingly.

4.2.5 I/O Summary

The *I/O Summary*, depicted in Figure 4.21, is a statistical chart giving an overview of the input-/output operations recorded in the trace file.

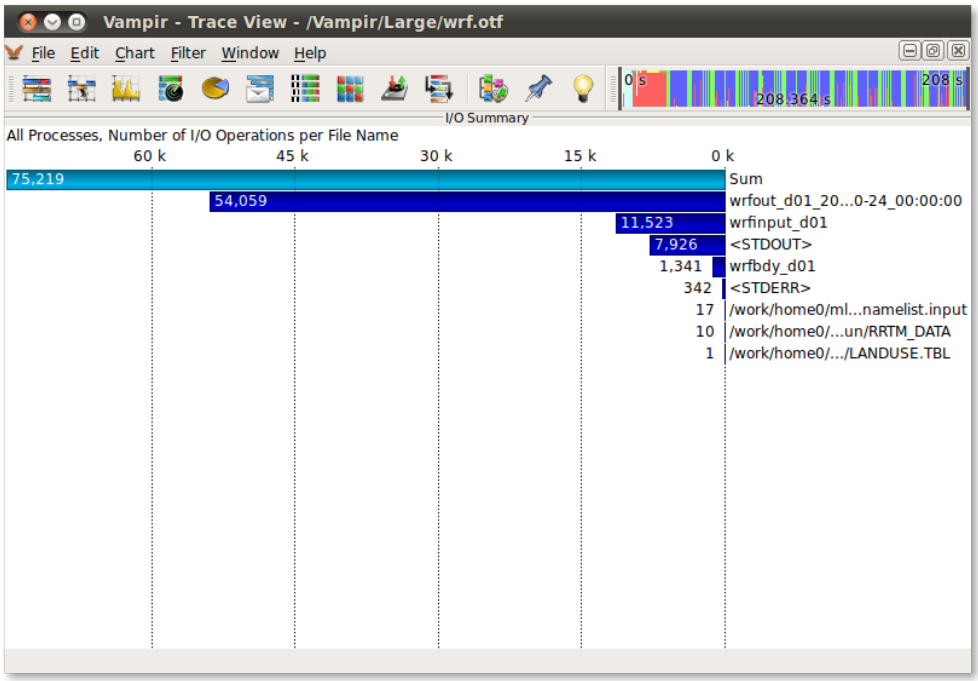


Figure 4.21: I/O Summary

All values are represented in a histogram like fashion. The text label indicates the group base while the number inside each bar represents the value of the chosen metric. The *Set Metric* sub-menu of the context menu is used to switch between the available metrics *Number of I/O Operations*, *Aggregated I/O Transaction Size*, *Aggregated I/O Transaction Time*, and values of *I/O Transaction Size*, *I/O Transaction Time*, or *I/O*

Bandwidth with respect to their selected value type. Therefore, one has the opportunity to switch between the value types *Minimum*, *Average*, *Maximum*, and *Average & Range* via the context menu entry *Set Value*.

Note: There will be one bar for every occurring metric. Furthermore, the value type *Average & Range* gives a quick and convenient overview and shows minimum, maximum, and average values at once. The minimum and maximum values are shown in an additional, smaller bar beneath the main bar indicating the average value. The additional bar starts at the minimum and ends at the maximum value of the metric, see Figure 4.19.

The I/O operations can be grouped by the characteristics *Transaction Size*, *File Name*, and *Operation Type*. The group base can be changed via the context menu entry *Group I/O Operations by*.

In order to select the I/O operation types that should be considered for the statistic calculation use the *Set I/O Operations* sub-menu of the context menu. Available options are *Read*, *Write*, *Read, Write*, and *Apply Global I/O Operations Filter*. The latter includes all selected operation types from the *I/O Events* filter dialog, see Section 5.3.

4.2.6 Call Tree

The *Call Tree*, depicted in Figure 4.22, illustrates the invocation hierarchy of all monitored functions in a tree representation. The display reveals information about the number of invocations of a given function, the time spent in the different calls and the caller-callee relationship.

The entries of the *Call Tree* can be sorted in various ways. Simply click on one header of the tree representation to use its characteristic to re-sort the *Call Tree*. Please note that not all available characteristics are enabled by default. To add or remove characteristics use the *Set Metric* sub-menu of the context menu.

To leaf through the different function calls, it is possible to fold and unfold the levels of the tree. This can be achieved by double clicking a level, or by using the fold level buttons next to the function name.

Functions can be called by many different caller functions, what is hardly obvious in the tree representation. Therefore, a relation view shows all callers and callees of the currently selected function in two separated lists, shown in the lower area in Figure 4.22.

In order to find a certain function by its name, Vampir provides a search option accessible via the context menu entry *Find...*. The entered keyword has to be confirmed by pressing the *Return* key. The *Previous* and *Next* buttons can be used to flip through the results.

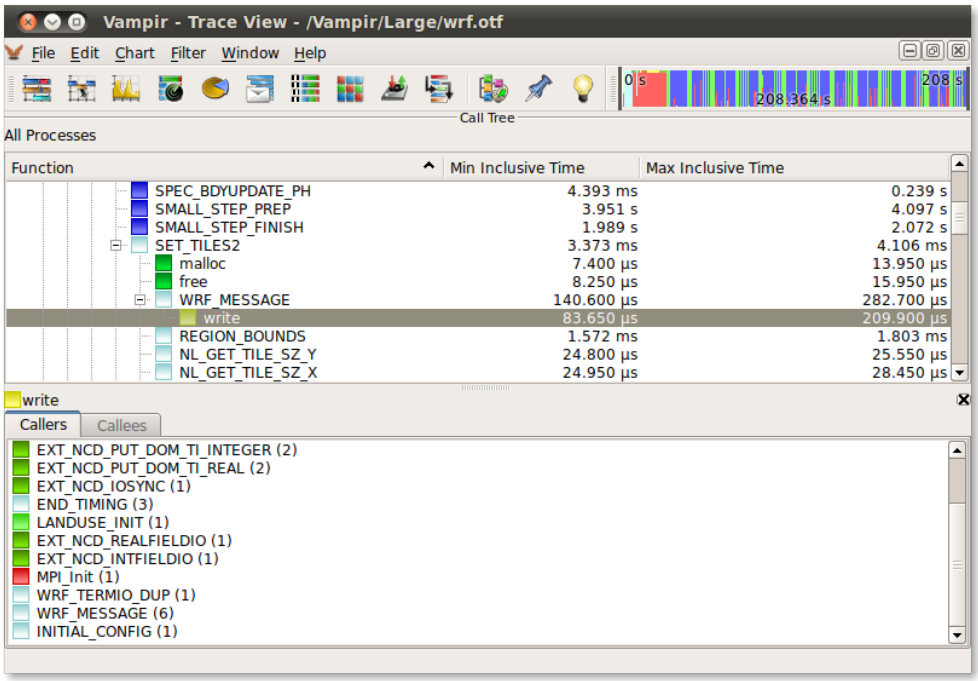


Figure 4.22: Call Tree

4.3 Informational Charts

4.3.1 Function Legend

The *Function Legend* lists all visible function groups of the loaded trace file along with their corresponding color.

If colors of functions are changed, they appear in a tree like fashion under their respective function group as well, see Figure 4.23.

4.3.2 Marker View

The *Marker View* lists all marker events included in the trace file.

The display organizes the marker events based on their respective groups and types in a tree like fashion. Additional information like the time of occurrence or descriptions are provided for each marker.

By clicking on a marker event in the *Marker View* this event becomes selected in the timeline displays. If this marker is located outside the visible area the zoom jumps to this event automatically. It is possible to select marker events by their type as well. Then all events belonging to that type are selected in the *Master Timeline* and the *Process Timeline*. By holding the *Ctrl* or *Shift* key pressed multiple marker events can

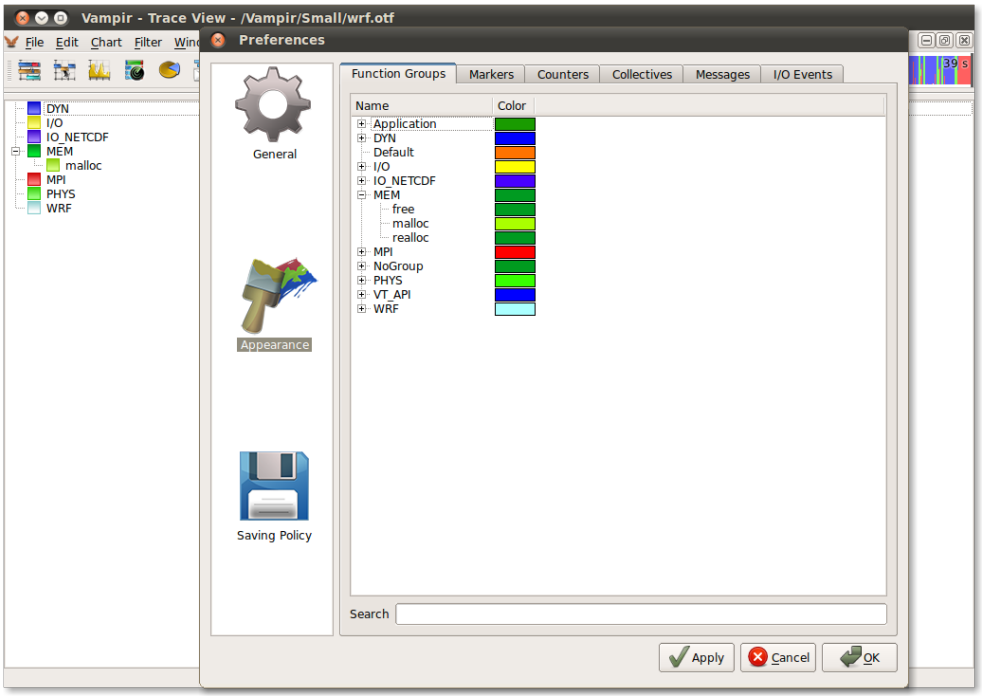


Figure 4.23: Function Legend

be selected. If exactly two marker events are selected the zoom is set automatically to the occurrence time of the markers.

4.3.3 Context View

As implied by its name, the *Context View* provides detailed information of a selected object additional to its graphical representation.

An object, e.g., a function, function group, message, or message burst can be selected directly in a chart by clicking its graphical representation. For different types of objects different context information is provided in the *Context View*. For example the object specific information for functions includes properties like *Interval Begin*, *Interval End*, and *Duration*, shown in Figure 4.25. Objects may provide additional information for some items. In that case such items are displayed as links. A click (double-click on OS X systems) on the link opens a new tab containing the additional information.

The *Context View* may contain several tabs. A new empty tab can be added by clicking the *+*-symbol on the right hand side. Information of new selected objects are always displayed in the currently active tab.

The *Context View* offers a mode for the comparison of information between tabs. The *=*-button on the left hand side allows to choose two objects for comparison. It is pos-

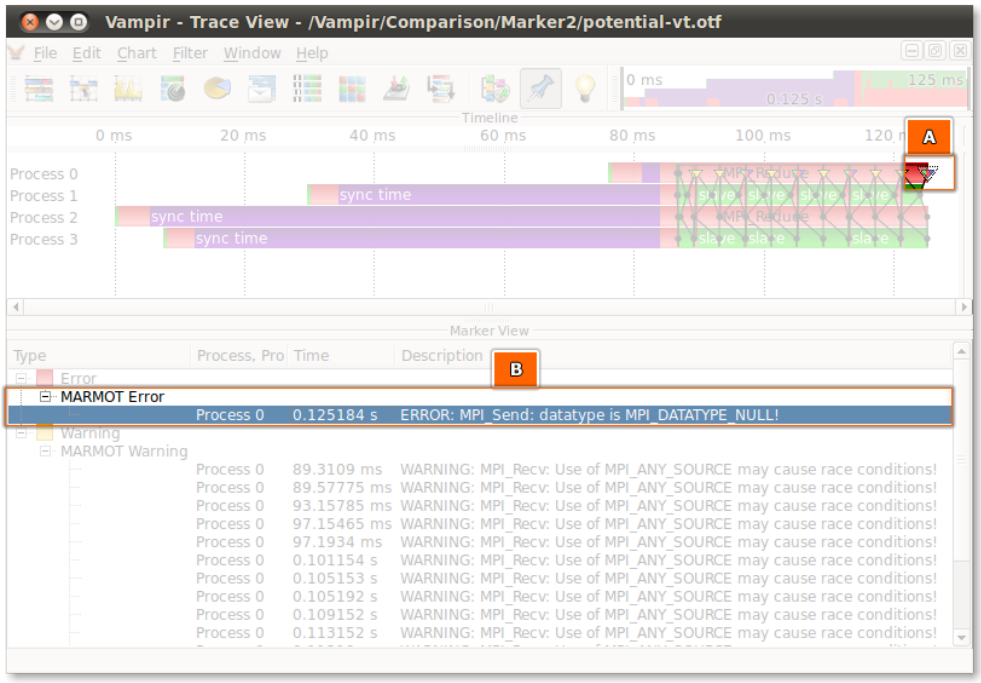

Vampir - Trace View - /Vampir/Comparison/Marker2/potential-vt.otf
File Edit Chart Filter Window Help
Timeline
0 ms 20 ms 40 ms 60 ms 80 ms 100 ms
A
Process 0
Process 1
Process 2
Process 3
sync time
sync time
sync time
Marker View
Type Process, Pro Time Description
B
Error
MARMOT Error
Process 0 0.125184 s ERROR: MPI_Send: datatype is MPI_DATATYPE_NULL!
Warning
MARMOT Warning
Process 0 89.3109 ms WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 89.57775 ms WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 93.15785 ms WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 97.15465 ms WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 97.1934 ms WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 0.101154 s WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 0.105153 s WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 0.105192 s WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 0.109152 s WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
Process 0 0.113152 s WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!

Figure 4.24: A chosen marker (A) and its representation in the Marker View (B)

sible to compare different objects from different charts. This might be useful in some analysis cases. The comparison shows a list of common properties along with the corresponding values. Differences are displayed as well. The first line always indicates the names of the respective charts, see Figure 4.26.

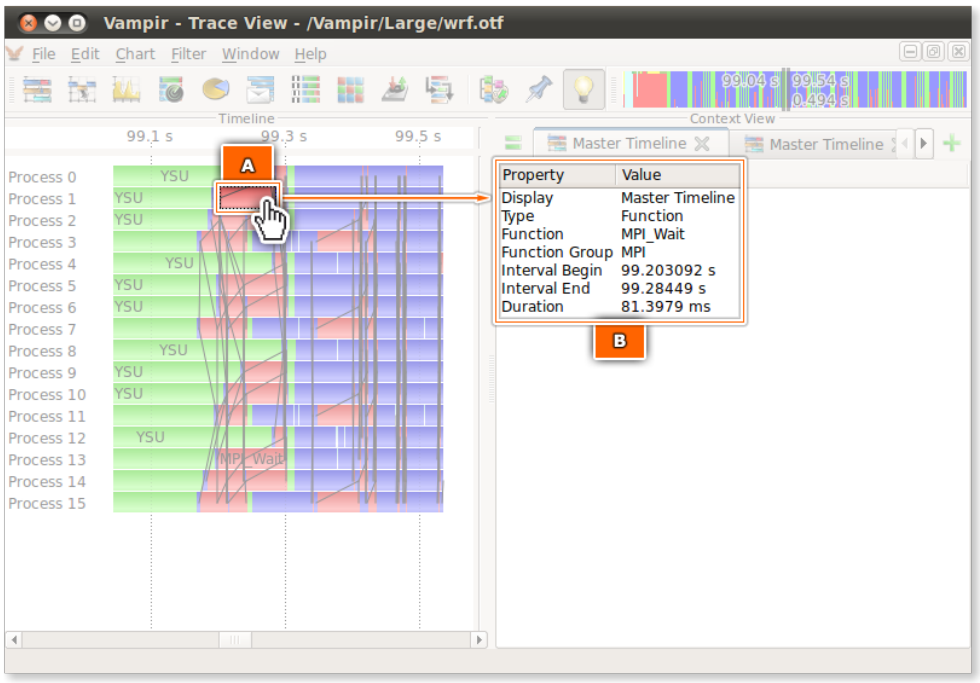


Figure 4.25: Context View, showing context information (B) of a selected function (A)

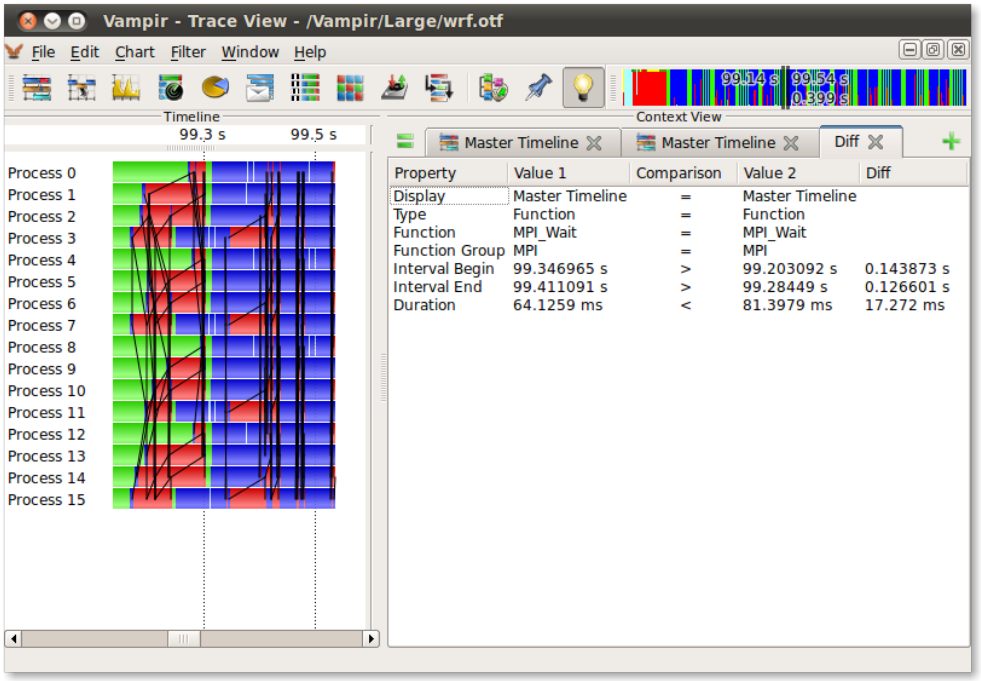


Figure 4.26: Comparison between Context Information

4.4 Customizable Performance Metrics

Vampir is shipped with a set of predefined customizable metrics that reflect known sources for performance issues and can serve as starting point for application specific customizations. Figure 4.27 shows the list of custom metrics that are predefined in Vampir. The list is accessible via the context menu entry *Customize Metrics...* in the Performance Radar or the Counter Data Timeline chart.

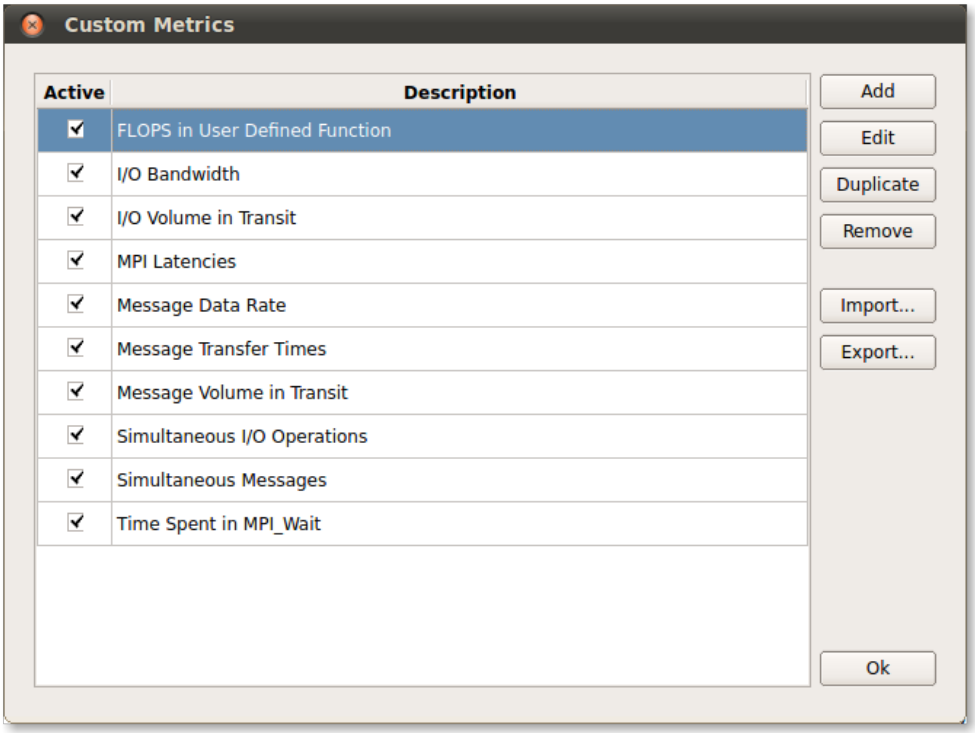


Figure 4.27: List of predefined customizable performance metrics

The following time dependent metrics are provided:

- *FLOPS in User Defined Function:* Floating point performance for a given function, which can be set by the user (see Section 4.4.1)
- *I/O Bandwidth:* Aggregated file I/O bandwidth (requires that I/O events have been recorded)
- *I/O Volume in Transit:* Aggregated number of bytes in transit to and from the I/O system
- *MPI Latencies:* Duration of individual MPI calls
- *Message Data Rate:* Bytes per second exchanged with message passing directives

- *Message Transfer Times:* Latencies of individual message passing directives
- *Message Volume in Transit:* Aggregated number of bytes in transit via messages
- *Simultaneous I/O Operations:* Number of interleaved I/O directives
- *Simultaneous Messages:* Number of interleaved message passing directives
- *Time Spent in MPI_Wait:* Times spent in MPI_Wait routines

4.4.1 Metric Editor

The *Custom Metrics Editor* allows to define derived metrics based on existing counters and functions. This is particularly useful as the performance data overlay of the Master Timeline, Section 4.1.3, is capable of displaying such custom metrics as well. The editor is accessible via the list of customizable performance metrics explained in the previous section by clicking on the *Edit* button. Figure 4.28 shows an example construction of a custom metric *Wait Time*. This metric is an addition of the time spent in the functions `MPI_Irecv` and `MPI_Wait`. Custom metrics are build from input metrics that are linked together using a set of available operations. In the editor the context menu, accessible via the right mouse button, allows to add new input metrics and operations. All created custom metrics become available in the *Set Metric* selections of the Performance Radar and Counter Data Timeline charts. They are available as well in the overlay mode of the Master Timeline. Custom metrics can be exported and imported in order to use them in multiple trace files.

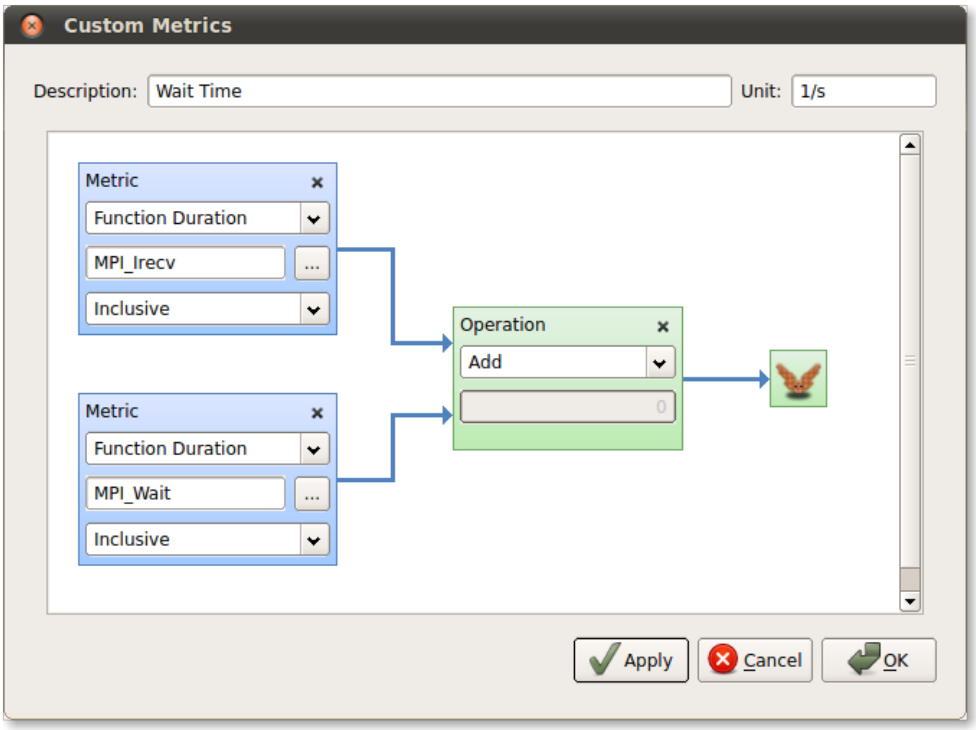


Figure 4.28: Custom metrics editor showing the construction of a custom *Wait Time* metric; The metric is defined by the addition of the duration of `MPI_Irecv` and `MPI_Wait` functions

4.4.2 Examples

MPI_Wait Duration

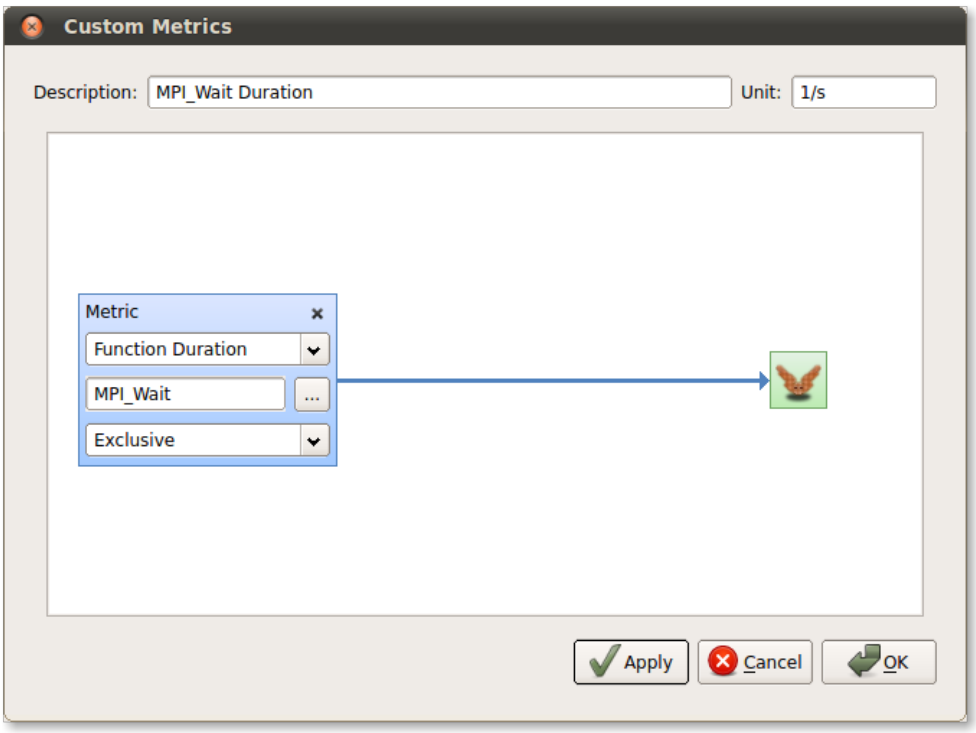


Figure 4.29: Construction of a custom metric showing the `MPI_Wait` duration

In Vampir it is also possible to identify long running functions. In this example long running invocations of the function `MPI_Wait` are highlighted.

First step is to construct a custom metric showing the `MPI_Wait` duration time. The custom metric editor is described in more detail in Section 4.4. The constructed custom metric is depicted in Figure 4.29.

Then the performance data overlay is used to show the own metric in the Master Timeline, Figure 4.30. The color scale is configured to show only `MPI_Wait` invocations with a high duration. After identification of the areas with the highest duration (deep red), zooming into such an area will eventually reveal the respective `MPI_Wait` invocations. Using the opacity slider, Figure 4.31, the individual function occurrences become visible in the Master Timeline.

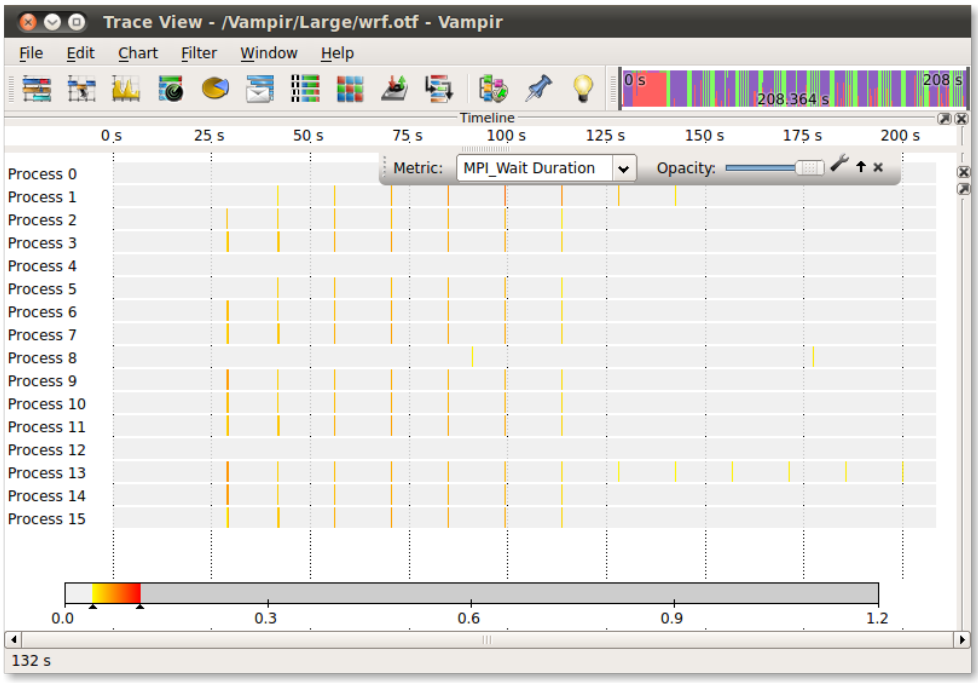


Figure 4.30: `MPI_Wait` invocations with longest duration

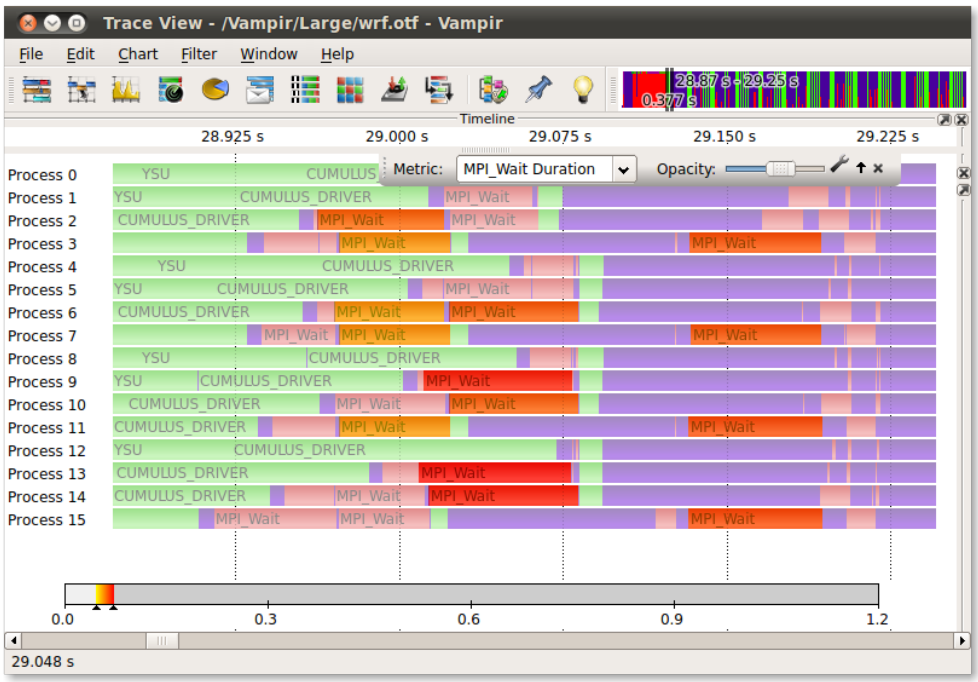


Figure 4.31: Using the opacity slider to reveal `MPI_Wait` invocations in the timeline together with the superimposed, color-coded duration

FLOPS of SOLVE_EM

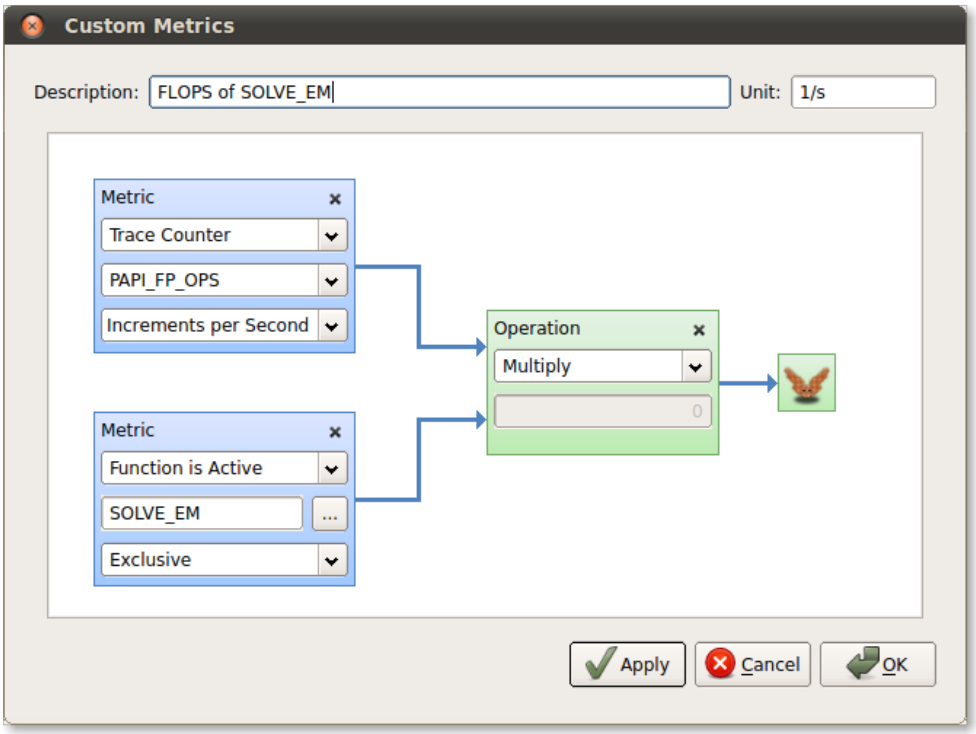


Figure 4.32: Custom metric showing FLOPS only for function `SOLVE_EM`

Vampir also allows to search for invocations of individual functions below or above a certain threshold. In this example invocations of the function `SOLVE_EM` with a FLOP rate above 150 M are searched.

Therefore the first step is to construct a custom metric showing the FLOP rate only for the function `SOLVE_EM`. The process of constructing a custom metric is described in more detail in Section 4.4. The constructed custom metric is depicted in Figure 4.32.

Figure 4.33 shows the constructed metric in the overlay. The color scale is set to highlight only functions above 150 M FLOPS. When zooming into an area of interest the opacity slider can be used to reveal individual function invocations in the timeline, Figure 4.34.

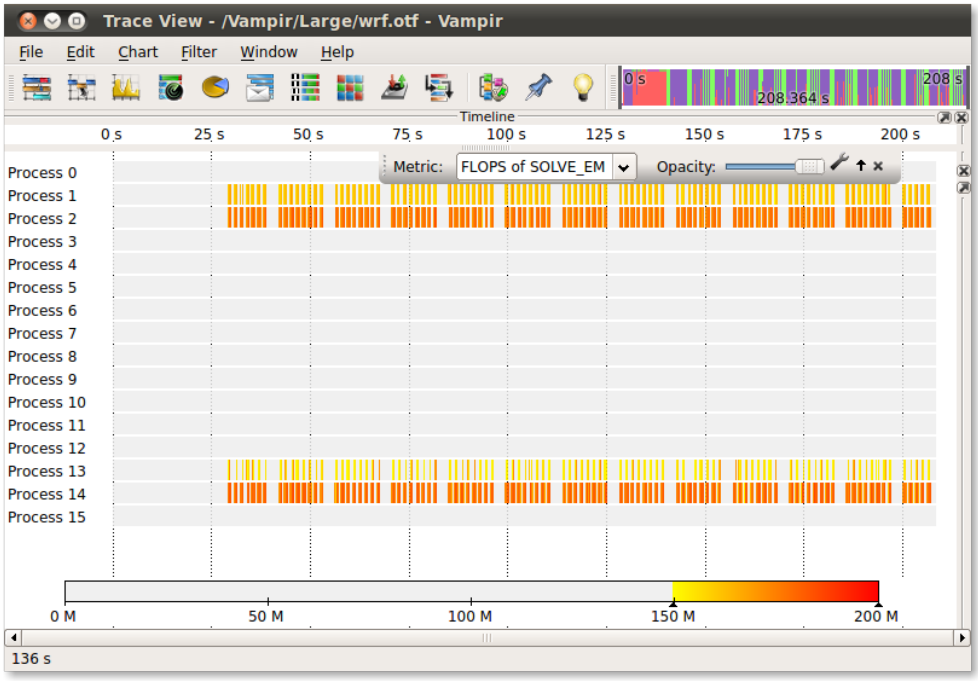


Figure 4.33: `SOLVE_EM` invocations with highest FLOP rate

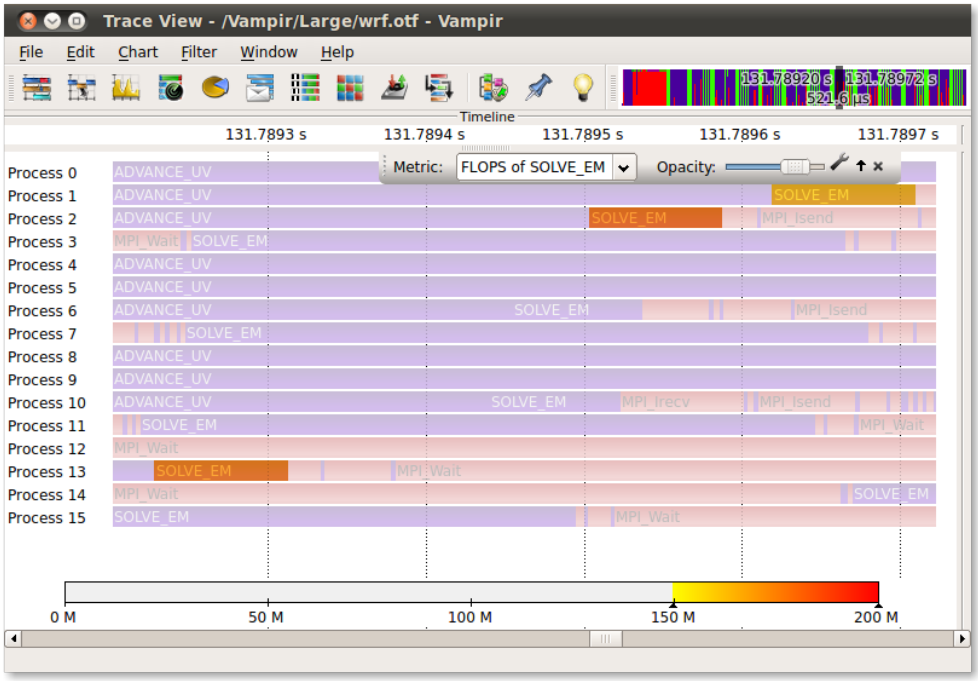

Trace View - /Vampir/Large/wrf.otf - Vampir
File Edit Chart Filter Window Help
131.78920 s
131.78972 s
521.6 µs
Timeline
131.7893 s 131.7894 s 131.7895 s 131.7896 s 131.7897 s
Metric: FLOPS of SOLVE_EM
Opacity:
Process 0 ADVANCE_UV
Process 1 ADVANCE_UV SOLVE_EM
Process 2 ADVANCE_UV SOLVE_EM MPI_Isend
Process 3 MPI_Wait SOLVE_EM
Process 4 ADVANCE_UV
Process 5 ADVANCE_UV
Process 6 ADVANCE_UV SOLVE_EM MPI_Isend
Process 7 SOLVE_EM
Process 8 ADVANCE_UV
Process 9 ADVANCE_UV
Process 10 ADVANCE_UV SOLVE_EM MPI_Irecv MPI_Isend
Process 11 SOLVE_EM MPI_Wait
Process 12 MPI_Wait
Process 13 SOLVE_EM MPI_Wait
Process 14 MPI_Wait SOLVE_EM
Process 15 SOLVE_EM MPI_Wait
0 M 50 M 100 M 150 M 200 M

Figure 4.34: Using the opacity slider to investigate individual invocations of `SOLVE_EM`

5 Information Filtering and Reduction

Due to the large amount of information that can be stored in trace files, it is usually necessary to reduce the displayed information according to some filter criteria. In Vampir, there are different ways of filtering. It is possible to limit the displayed information to a certain choice of processes or to specific types of communication events, e.g. to certain types of messages or collective operations. Deselecting an item in a filter means that this item is fully masked. In Vampir, filters are global. Therefore, masked items will no longer show up in any chart. Filtering not only affects all performance charts, but also the *Zoom Toolbar*. All filter can be reached via the *Filter* entry in the main menu.

The available filter and their respective filter criteria are summarized in Table 5.1.

Filtered Object	**Filter Criteria**
Processes	Process Groups Communicators Process Hierarchy Representative Processes
Messages	Message Communicators Message Tags
Functions	Name Duration Number of Invocations
Collective Operations	Communicators Collective Operations
I/O Events	I/O Groups File Names Operation Types

Table 5.1: Object Filtering Options

Note: The available selection methods are the same across all filter dialogs (except the Function Filter). The check box *Include/Exclude All* either selects or deselects every item. Specific items can be selected/deselected by clicking into the check box next to it. Furthermore, it is possible to select/deselect multiple items at once. For this, mark the desired entries by clicking their names while holding either the *Shift* or the *Ctrl* key. By holding the *Shift* key every item between the two clicked items will be marked. Holding

the *Ctrl* key, on the other hand, enables you to add or remove specific items from/to the marked ones. Clicking into the check box of one of the marked entries will cause selection/deselection for all of them.

5.1 Process Filter

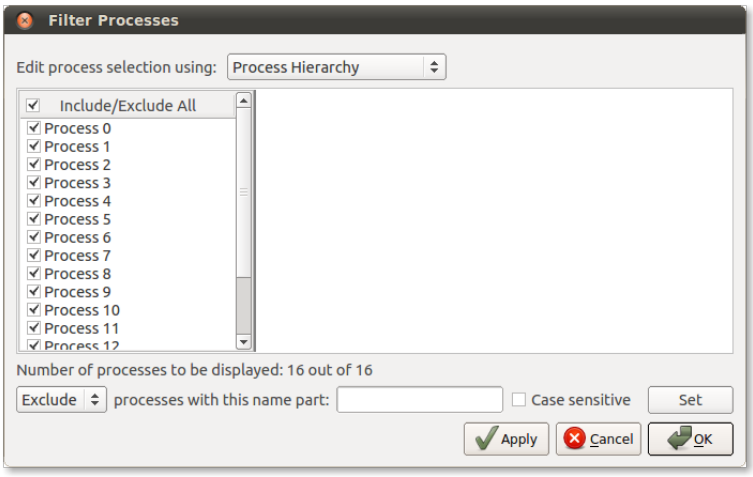


Figure 5.1: Process Filter

Figure 5.1 shows a typical process representation in the *Process Filter* dialog. Processes can be filtered by their *Process Hierarchy*, *Communicators*, *Process Group*, and *Representative Processes*. Items to be filtered are arranged in a spreadsheet representation. In addition to selecting or deselecting an entire group of processes, it is also possible to filter single processes.

You can also use regular expressions to filter processes. It is also possible to use a wildcard like ”Process*”. You can escape characters with a backslash. If no process name matches the given pattern, a simple string comparison will be performed.

Examples:

- `Process .*[13579]$` matches odd-numbered items
- `Process .*0{1}$` matches eyery tenth item
- `Process .*0{2}$` matches eyery hundredth item
- `Process` matches all containing ’Process’
- `^Process$` matches exactly ’Process’

5.2 Message and Collective Operations Filter

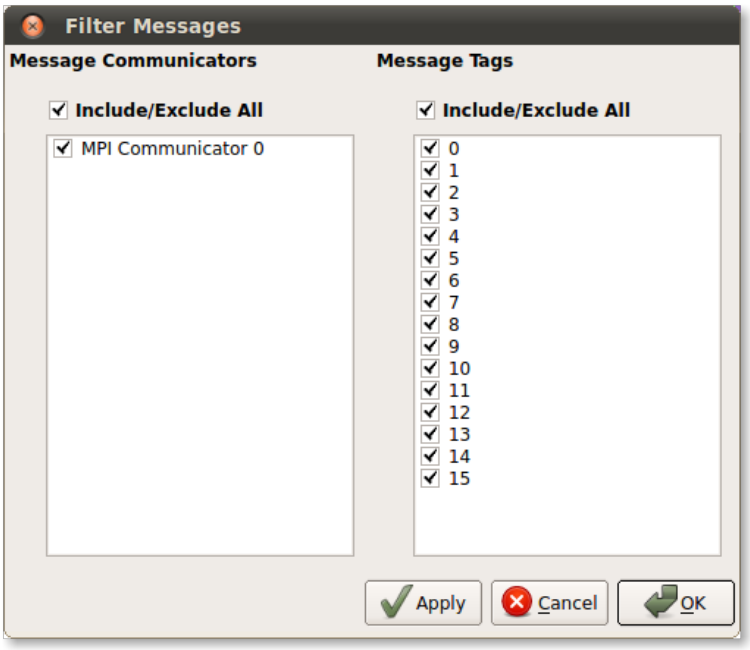


Figure 5.2: Message Filter

Figure 5.2 shows a *Message Filter* dialog. This dialog allows to filter messages from the displayed trace data. Available options are to select/deselect messages based on their *Message Tag* or *Message Communicator*. The default is to show all messages. The *Collectives Filter* is designed accordingly. It allows to filter collective operations from the displayed trace data. The collectives can be filtered by their *Communicator* or their *Collective Operation* type.

5.3 I/O Filter

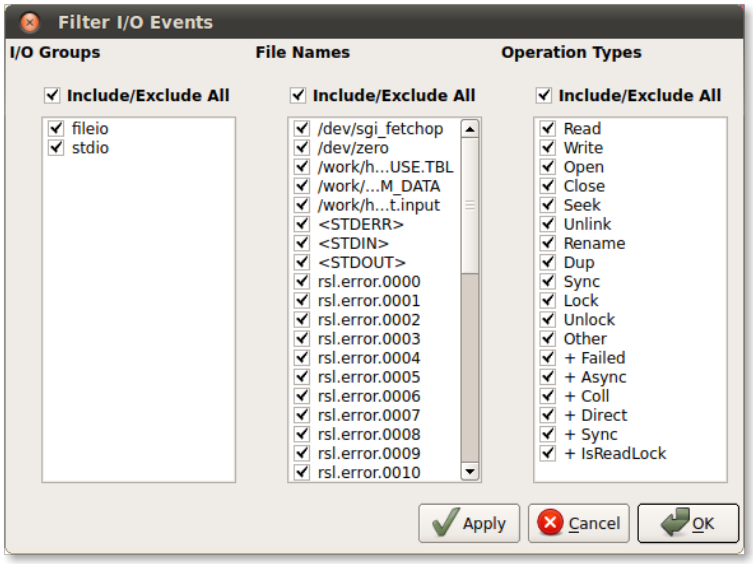


Figure 5.3: I/O Filter

Figure 5.3 depicts the *I/O Filter* dialog. The dialog allows to selectively filter I/O events displayed in timelines and statistics. Available filter criteria are *I/O Groups* or the I/O *Operation Type*. It is also possible to filter I/O operations based on input and output files.

5.4 Function Filter

The filtering of functions in Vampir is controlled via the *Function Filter Dialog*, which can be accessed via the main menu under *Filter* → *Functions...*. Initially, a list of available rule sets is depicted as can be seen in Figure 5.4. By default, the list only shows a *None* entry. It can only be one filter active at a given time. To select the active filter use the radio buttons on the left hand side of the list. Clicking on the *Add* button creates a new set of rules and shows the input mask depicted in Figure 5.5.

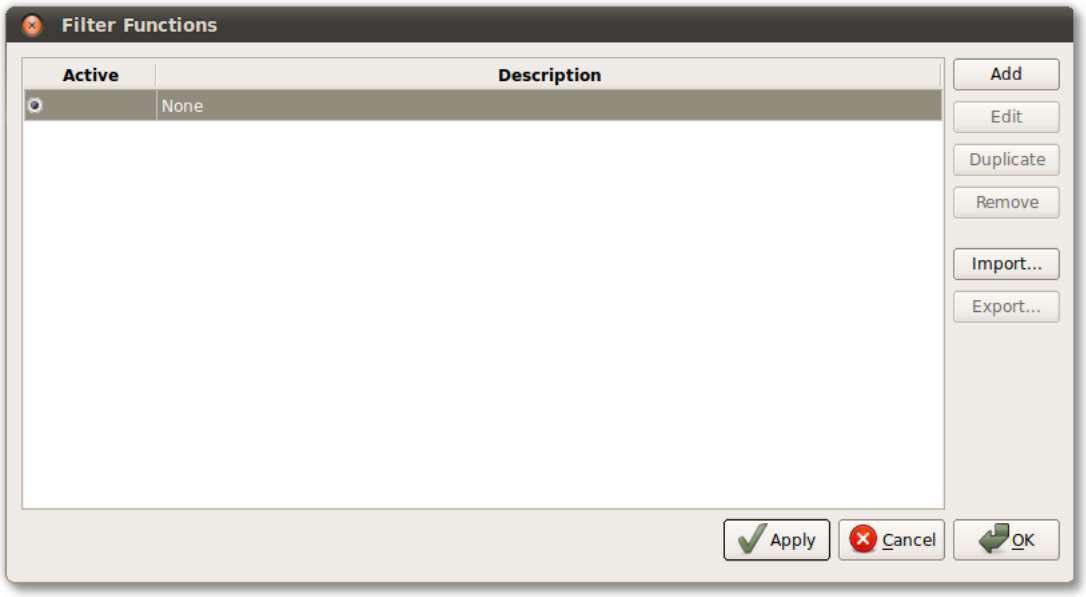


Figure 5.4: Function Filter Dialog with List of Rule Sets

The Function Filter Dialog is build on the concept of filter rules. The user can define several individual rules. The rules are explained in more detail in Chapter 5.4.1. The header of the dialog defines how multiple rules are evaluated. One possibility is to build up the filter in a way that combines the filter rules with an *and* relation. To choose this mode *all* must be selected in the combo box in the header of the dialog. This means that all rules must evaluate to true in order to produce the filter output. The other option is to combine the rules with an *or* relation. To choose this mode *any* must be selected in the combo box in the header of the dialog. In this case any rule must be evaluate to true in order to produce the filter output. The examples in Chapter 5.4.2 illustrate both modes.

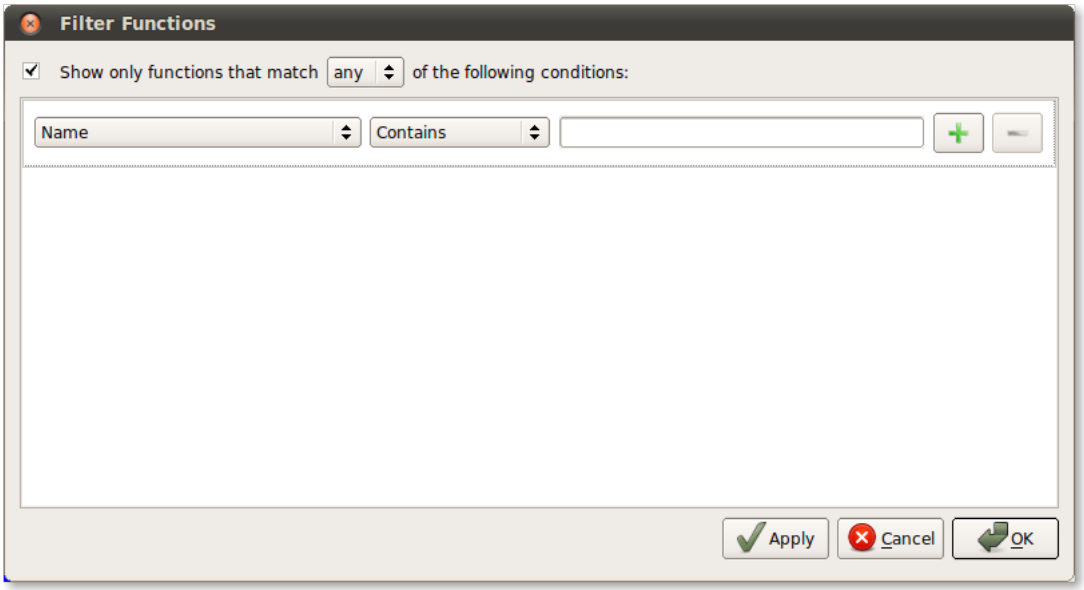


Figure 5.5: Function Filter Dialog with Rule Set

5.4.1 Filter Options

This chapter explains the various options available to build up filter rules.

Filtering Functions by Name

One way of filtering functions is by their name. This filter mode provides two different options.

Name provides a text field for an input string. Depending on the options, all functions whose names match the input string are shown. The matching is not case sensitive.

Available options:

- **Contains:** The given input string must occur in the function name.
- **Does not contain:** The given input string must not occur in the function name.
- **Is equal to:** The given input string must be the same as the function name.
- **Is not equal to:** The given input string must not be the same as the function name.
- **Begins with:** The function name must start with the given input string.
- **Ends with:** The function name must end with the given input string.

List of Names provides a dialog that allows to directly select the desired set of functions and function groups.

Available options:

- **Contains:** The selected functions are shown.
- **Does not contain:** The selected functions are filtered.

Filtering Functions by Duration

Functions can also be filtered by their duration. Duration of a function refers to the time spent in this function from the entry to the exit of the function.

There are two options available:

- **Is greater than:** All functions whose duration time is longer than the specified time are shown.
- **Is less than:** All functions whose duration time is shorter than the specified time are shown.

Filtering Functions by Number of Invocations

The number of invocations of a function can also be used as filter rule. This criteria refers to how often a function is executed in an application. There are two possible filter rules in this mode.

Number of Invocations shows functions based on their total number of invocations in the whole application run.

There are two options available:

- **Is greater than:** All functions whose number of invocations is greater than the specified number are shown.
- **Is less than:** All functions whose number of invocations is less than the specified number are shown.

Number of Invocations per Process shows functions based on their individual number of invocations per process. Hence, if the number of invocations of a function varies over different processes, this function might be shown for some processes and filtered for others.

There are two options available:

- **Is greater than:** All functions whose number of invocations is greater than the specified number are shown.

- **Is less than:** All functions whose number of invocations is less than the specified number are shown.

Filtering Functions by Call Path

The **Call Path** filter provides a string input field for a pattern. Depending on the options, all functions with their related events are shown which satisfy a substring match against the given pattern.

This filter mode provides two opposing options:

- **Contains:** The call path must contain a function where the given pattern must occur in the functions name. This specifically means that functions that lead to the matched function won't be shown anymore. The matched function itself along with its possibly called sub-functions is still shown. All other call paths that do not contain a matched function are filtered out as well and won't be shown.
- **Does not contain:** The call path must not contain a function where the given pattern occurs in the function name. This specifically means that only functions that lead to the matched function will be shown, excluding the matched function itself as well as its possibly called sub-functions. Call paths that do not contain a matched function are still shown and remain unaffected by the filter.

Filtering Functions by Call Level

Functions can also be filtered by their **Call Level**. This filter mode provides a number input field to select the call level.

Available options:

- **Is greater than:** All functions whose enter event is higher than the specified level are shown.
- **Is less than:** All functions whose enter event is lower than the specified level are shown.

5.4.2 Examples

In this chapter a few examples explain the usage of the function filter. This enables the user to understand the basic principles of function filtering in Vampir at a glace. It also illustrates a part of the set of available filter options provided by Vampir.

Unfiltered Trace File

This section introduces the example trace file in an unfiltered state. The timelines show a part of the initialization of the WRF weather forecast code. The red color corresponds to communication (MPI), whereas the purple areas represent some input functions of the weather model.

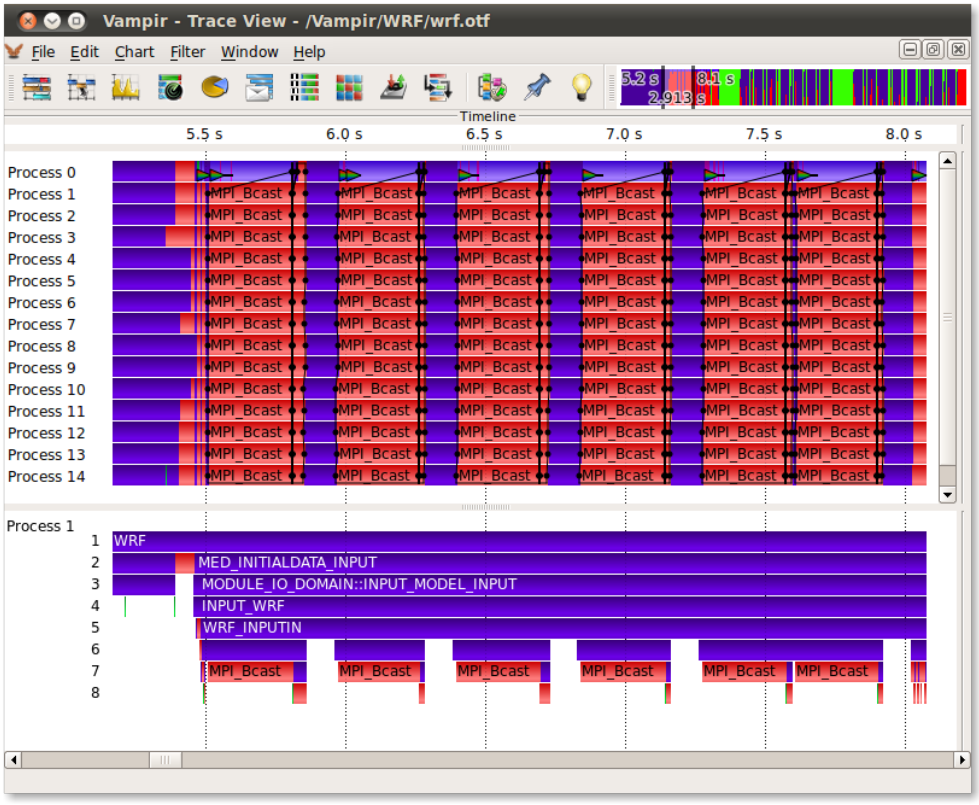


Figure 5.6: Master Timeline and Process Timeline without filtering

Showing only MPI Functions

In this example only functions that contain the string *mpi* (not case sensitive) somewhere in their name are shown. Since only MPI functions start with *MPI* in their name this filter setting shows all MPI functions and filters the others.

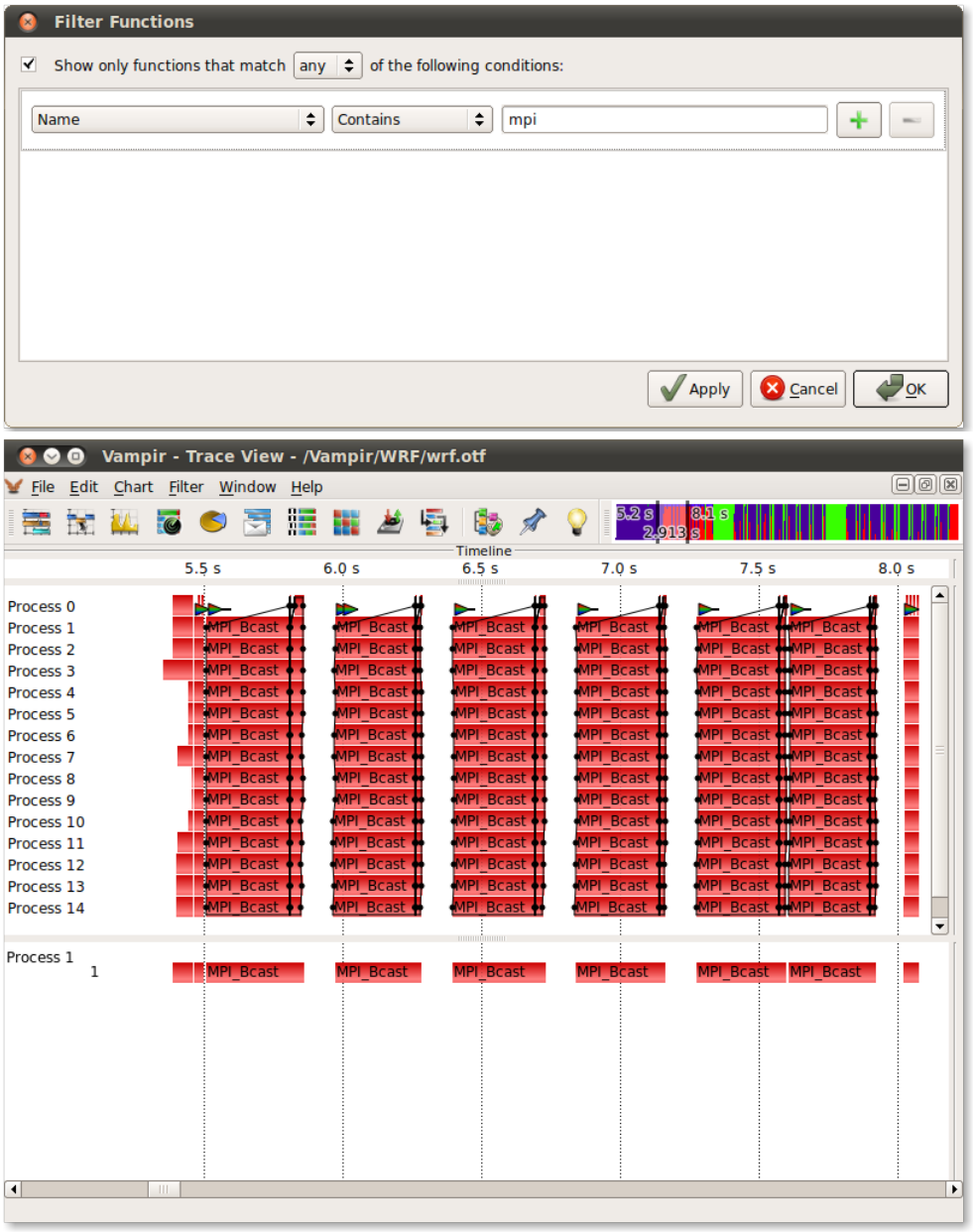


Figure 5.7: Showing only MPI

Showing only Functions with at least 250 ms Duration

This example demonstrates the filtering of functions by their duration. Here only long function occurrences with a minimum duration time of 250 ms are shown. All other functions are filtered.

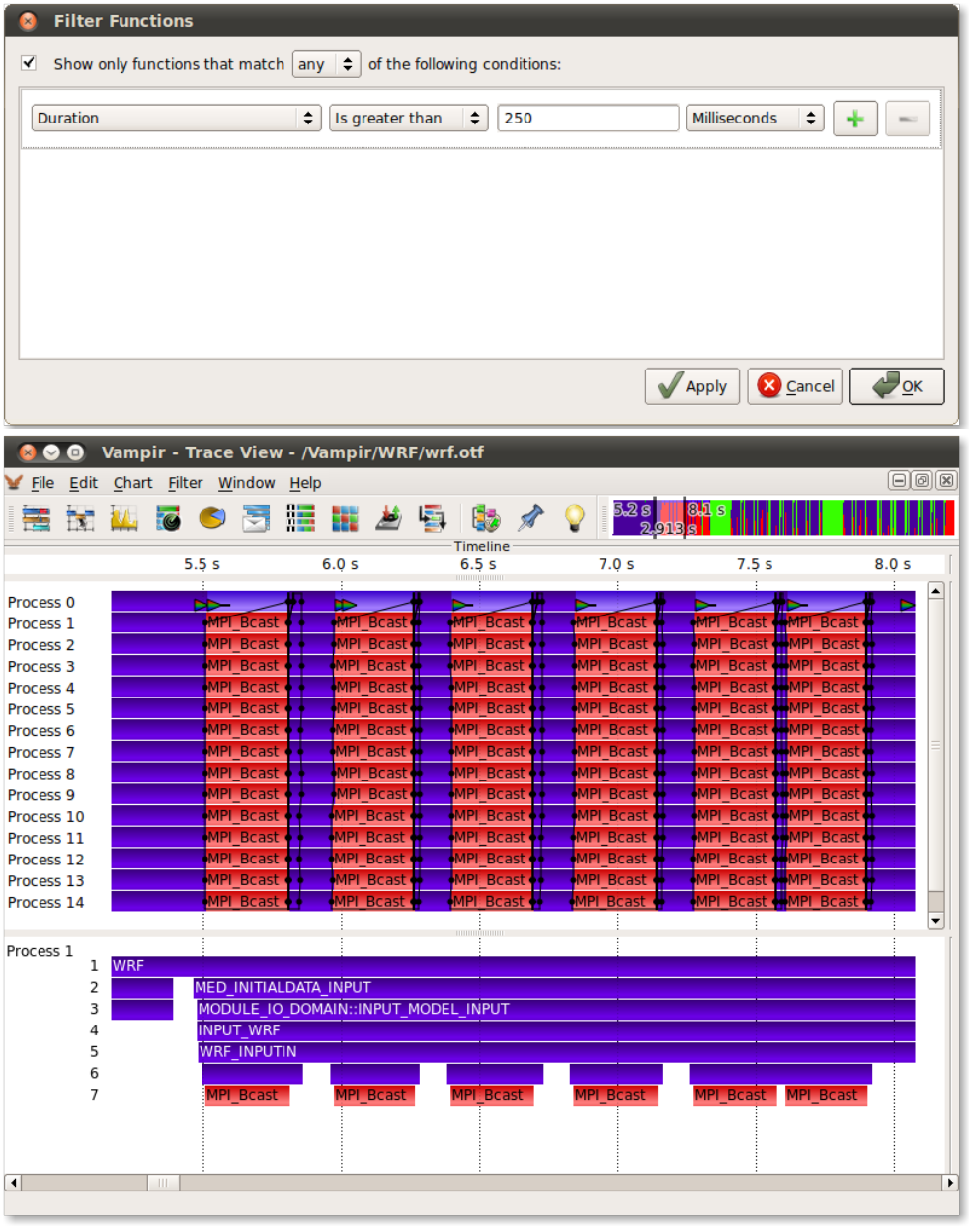


Figure 5.8: Showing only functions with more than 250 ms duration

Combining Function Name and Duration Rules

This example combines the two previous rules. First the *any* relation is used. Thus, the filter shows all functions that have at least 250 ms duration time and additionally also all *MPI* functions.

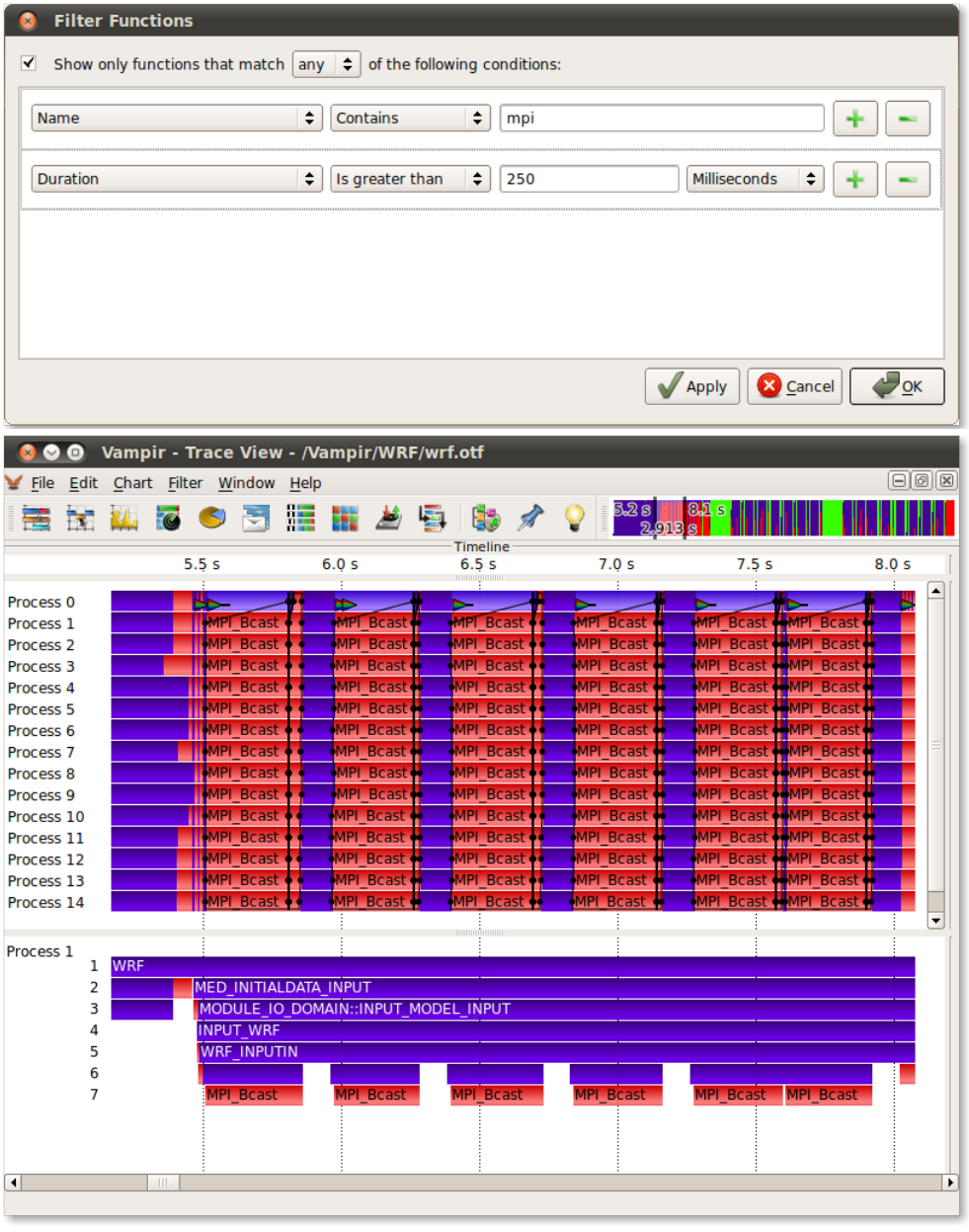


Figure 5.9: Combining rules using *any*

The second example illustrates the usage of the *all* relation. Here all shown functions have to satisfy both rules. Therefore the filter shows only *MPI* functions that have a duration time of more than 250 ms.

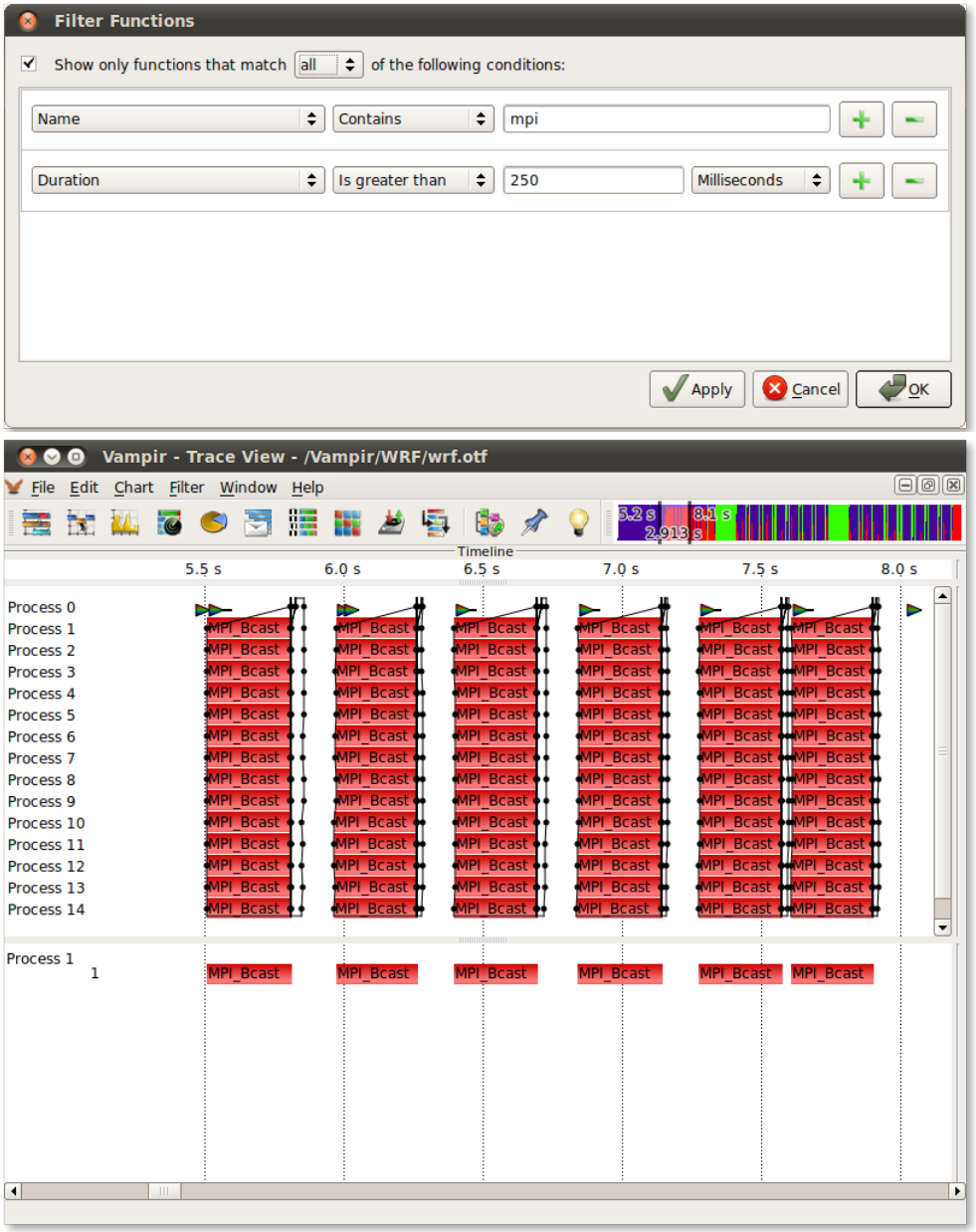


Figure 5.10: Combining rules using *all*

Building Ranges with Number of Invocation Rules

The combination of rules also allows for the filtering of functions in a specified criteria range. The following example filter setup shows all functions whose number of invocations lie inside the range between 2000 and 15000.

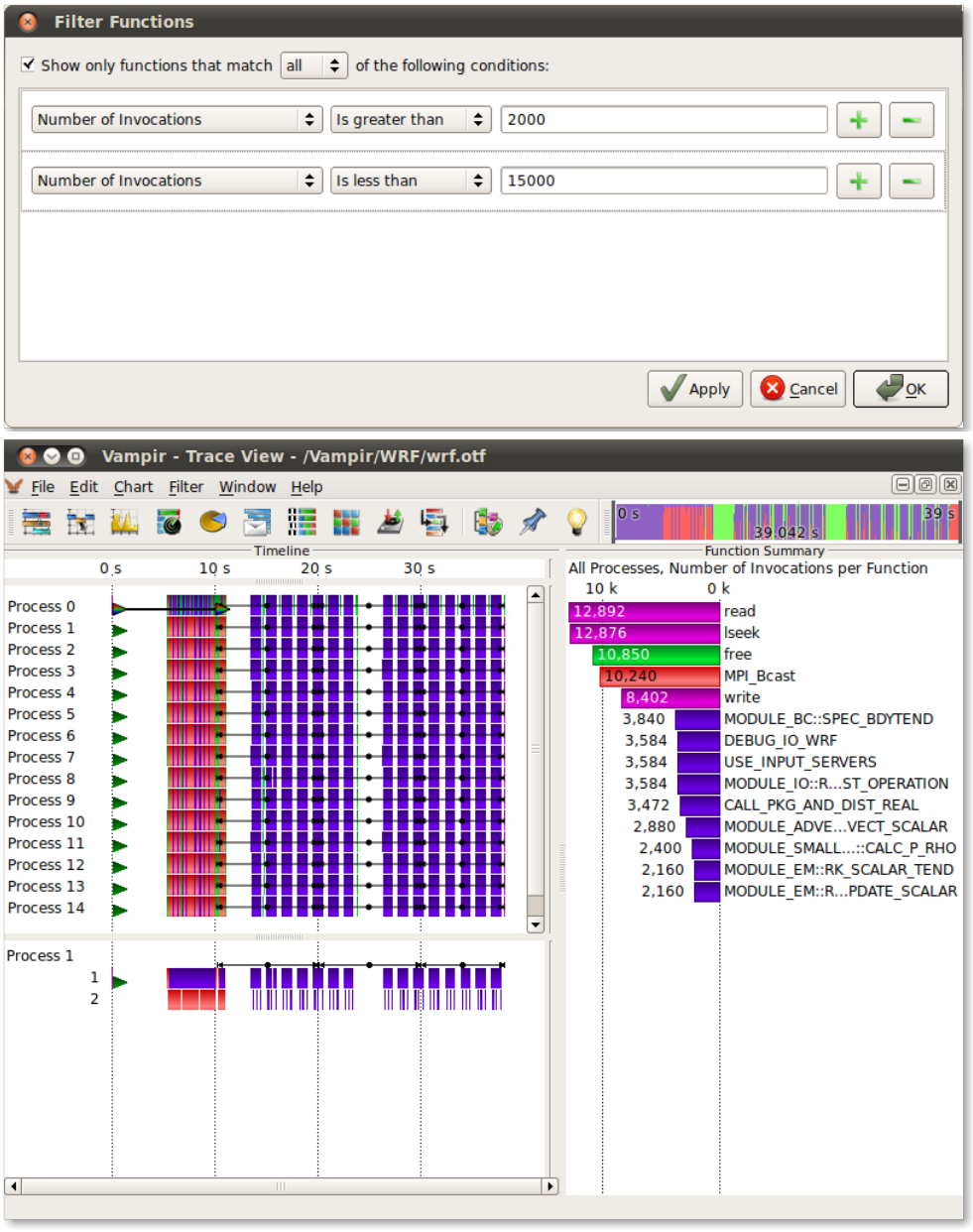


Figure 5.11: Show functions inside a specified range

This example demonstrates the opposite behavior of the previous example. Here all functions whose number of invocations lie outside the range between 2000 and 15000 are shown, i.e., functions with less than 2000 invocations and functions with more than 15000 invocations.

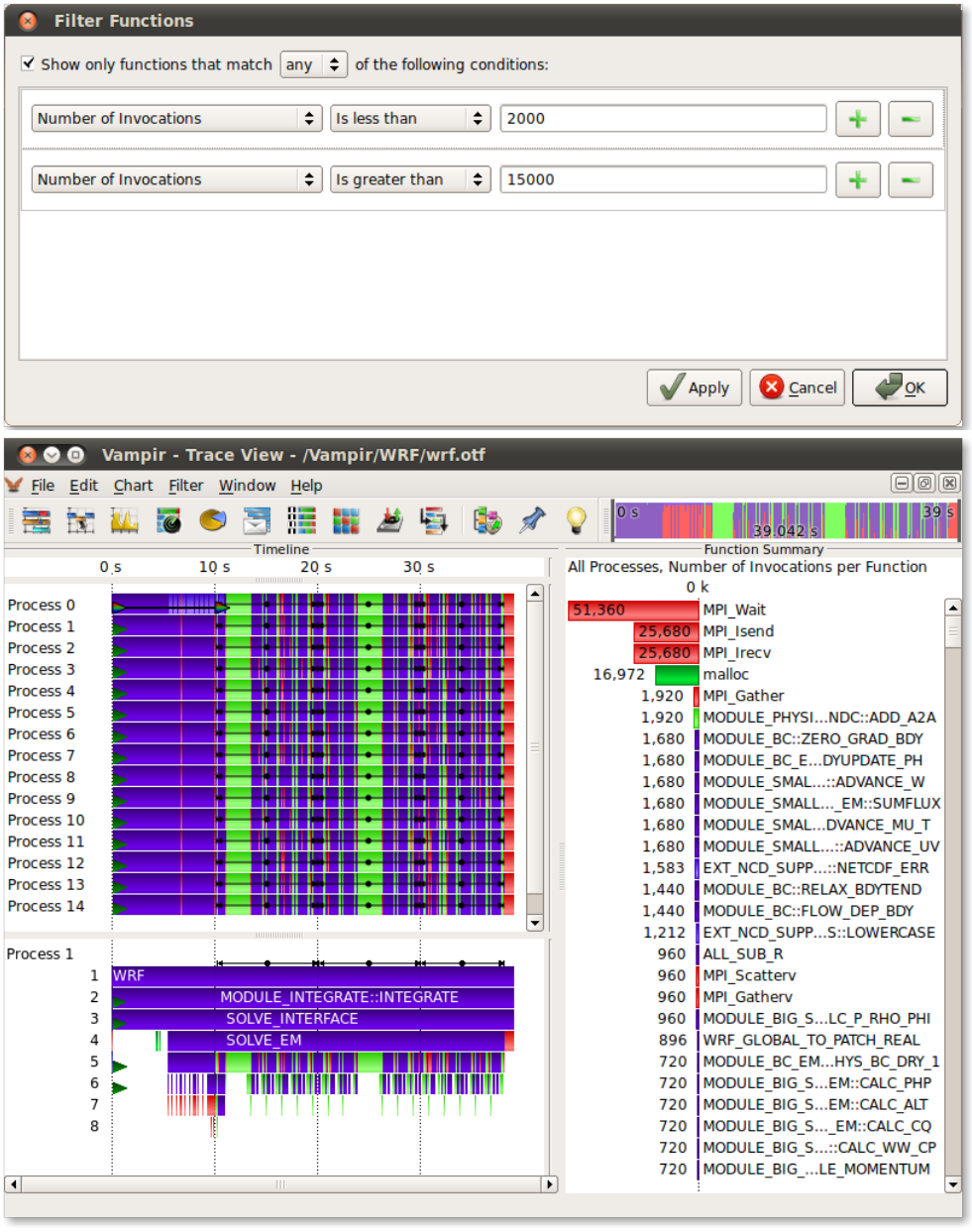


Figure 5.12: Show functions outside a specified range

Call Path contains WRF_INPUTIN

In this example only functions that are called, directly or indirectly, by `WRF_INPUTIN` are shown. As a consequence all call paths start with `WRF_INPUTIN`. All other functions are filtered.

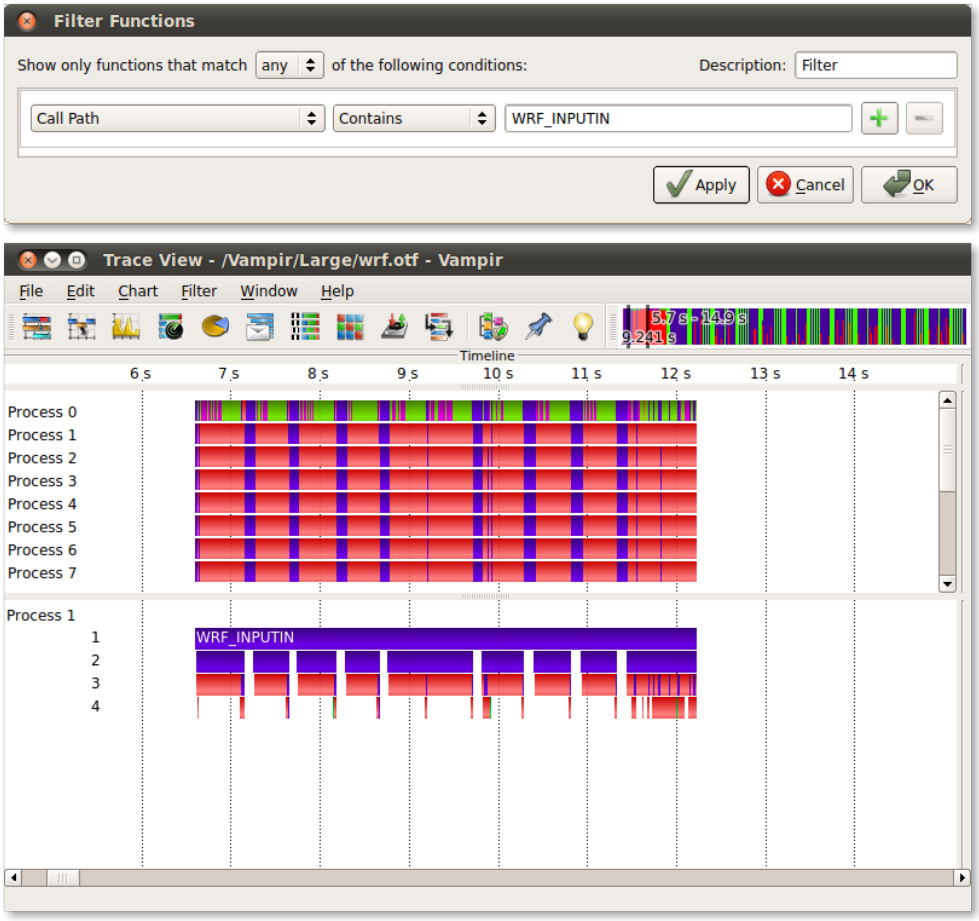


Figure 5.13: Call path filter which contains `WRF_INPUTIN`

Call Path does not contain WRF_INPUTIN

This example demonstrates the opposite behavior of the previous example. In call paths that contain the function `WRF_INPUTIN`, only functions that lead to `WRF_INPUTIN` are shown. The function `WRF_INPUTIN` itself and their, directly or indirectly, called sub-functions are filtered. Other call paths remain unaffected by the filter and are still shown.

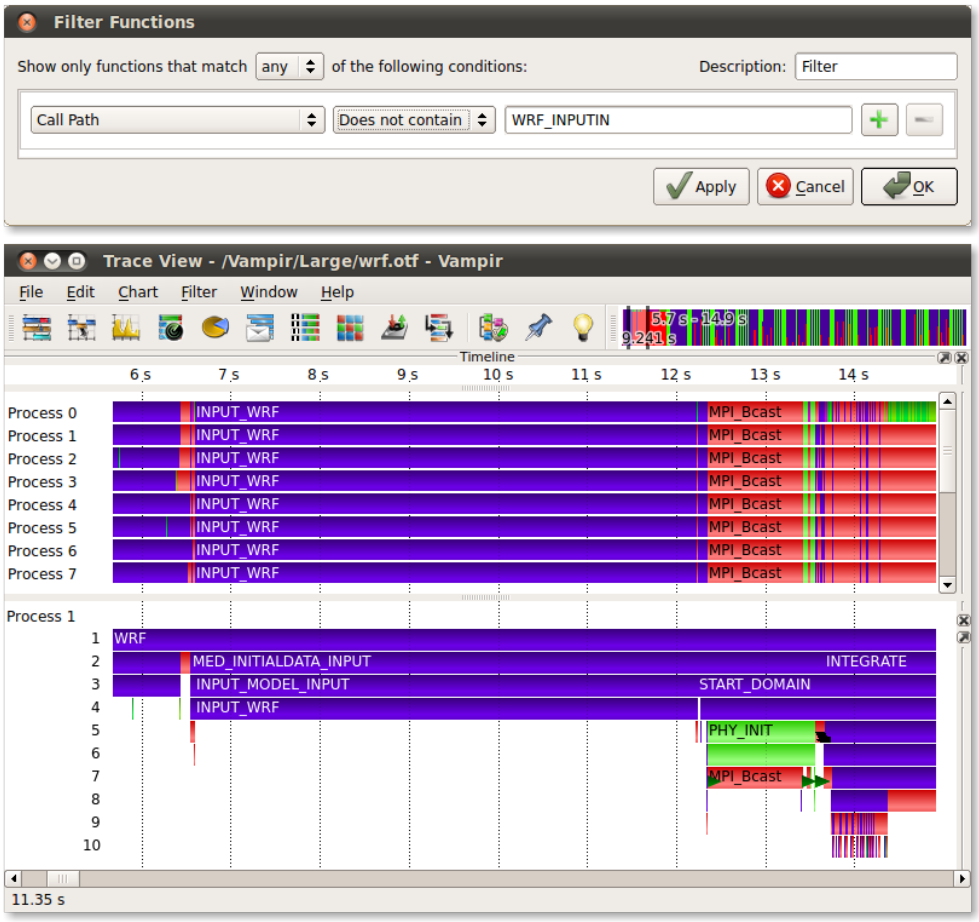


Figure 5.14: Call path filter which does not contain `WRF_INPUTIN`

Showing only Functions until a certain Call Level

This example demonstrates the filtering of functions by their call level. Here only functions with an enter event less then call level five are shown. All other functions are filtered.

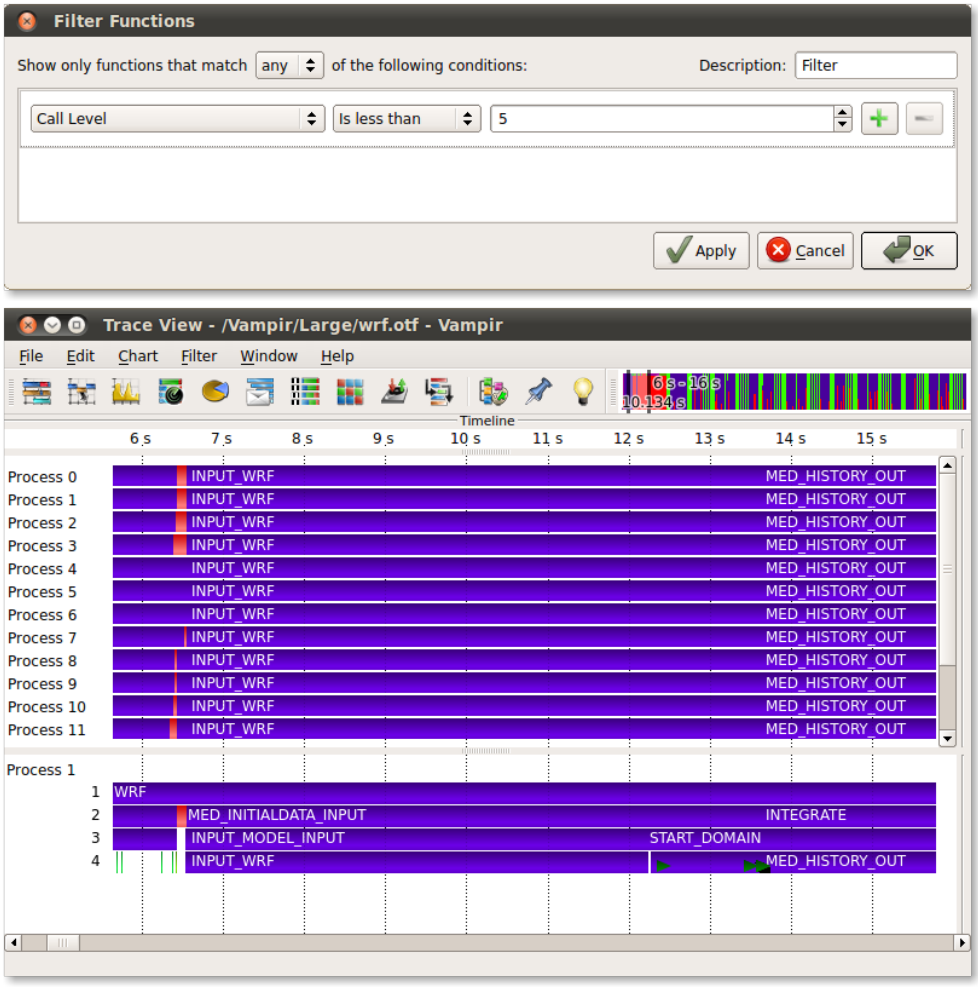


Figure 5.15: Showing only functions with a call level less than five

6 Comparison of Trace Files

In Vampir the comparison of trace files seamlessly integrates with the functionality explained in the previous chapters of this document. The user can benefit from already gained experiences. For the comparison of performance characteristics all common charts are provided. Additionally, in order to effectively compare multiple trace files, their zoom is coupled and synchronized. For the comparison of areas of interest the displayed trace regions are freely shiftable in time. This allows for arbitrary alignments of the trace files, and thus, enables comparison of user selected areas in the trace data.

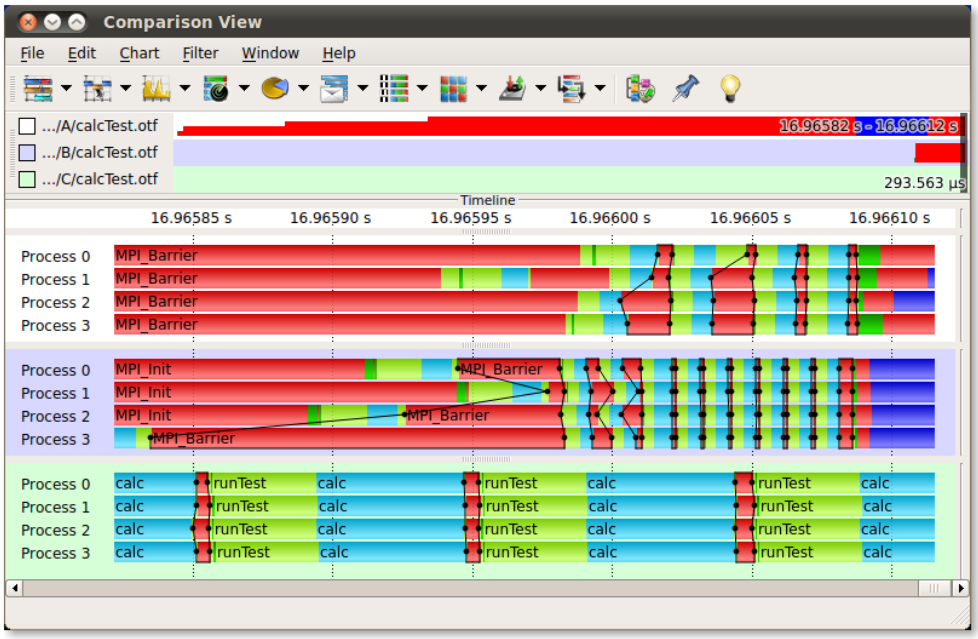


Figure 6.1: Comparison View

The *Comparison View* window, depicted in Figure 6.1, provides all comparison features. This chapter introduces its usage with the help of screenshots. For this purpose the comparison of three trace files is demonstrated step by step. The example trace files show one test application performing ten iterations of simple calculations. Each trace, respectively, represents the run of this application on a different machine.

6.1 Starting and Saving a Comparison Session

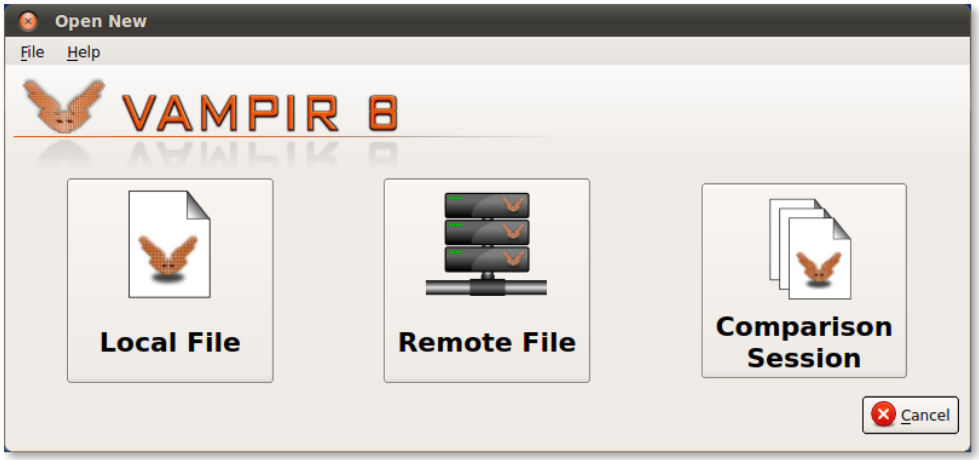


Figure 6.2: Vampir start window

The fist step in order to compare trace files in Vampir is to start a comparison session. A comparison session is setup using the *Comparison Session Manager*. This dialog is accessible via the main menu entry *File* → *Comparison Session Manager...* or by clicking the *Open Other...* button in the Vampir start window, Figure 6.2. The Comparison Session Manager, depicted in Figure 6.3, holds a list of trace files to be compared in the current session. The list is editable at any time using the plus and minus buttons. Clicking the *OK* button will load the respective trace files and open the Comparison View.

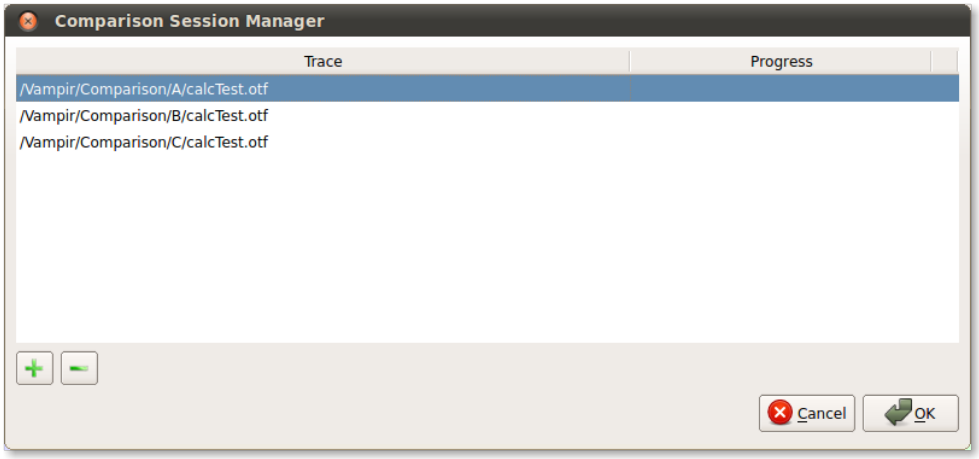


Figure 6.3: Comparison Session Manager listing three trace files for comparison

Figure 6.4 shows the resulting Comparison View. As indicated by the navigation toolbars at the top of the figure, all selected trace files are now included in a single Comparison View instance. The files in the view are sharing a coupled zoom. The usage of charts and zooming in this view is described in the next section.

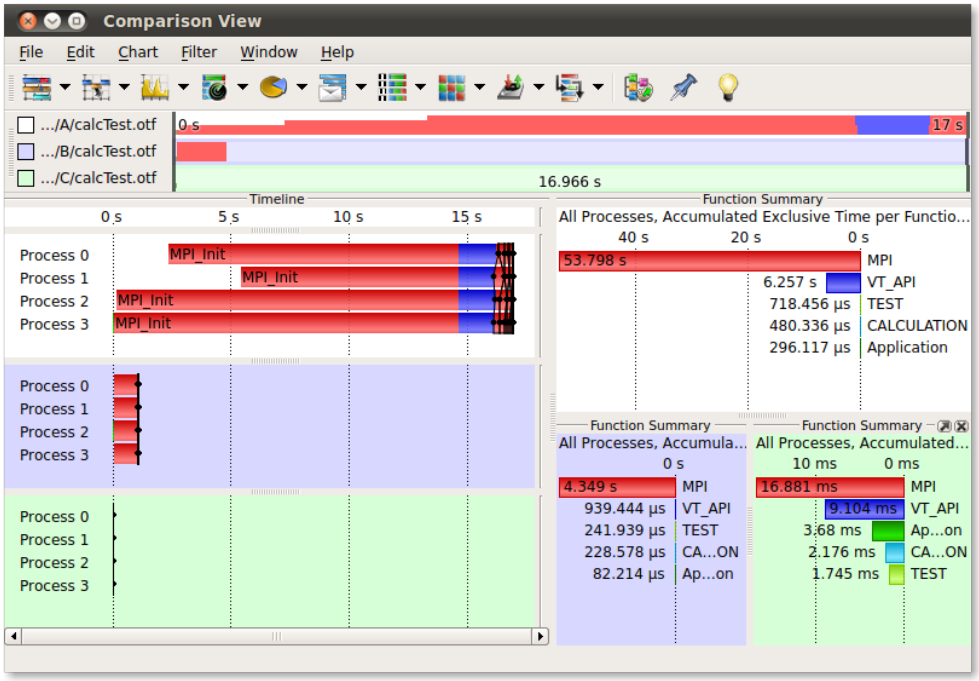


Figure 6.4: Open Comparison View

To save a comparison session use the menu entries *File* → *Save* or *File* → *Save As...*. This will store a **.vcompare* file containing the compared trace files, settings, and the Comparison View layout. To restore a comparison session simply open the respective *.vcompare file. Previous comparison sessions are also available in the recent open files list of Vampir.

6.2 Usage of Charts

For the comparison of performance metrics the Comparison View provides all common charts of Vampir. In contrast to the ordinary Trace View the Comparison View opens one chart instance for each trace file, i.e., with three open trace files, one click on the Master Timeline icon opens three Master Timeline charts. By using the icon menus, accessible via the triangles next to the chart icons, it is also possible to open only one chart instance for the selected trace. Also, in order to distinguish the same charts between the trace files, a dedicated background color is assigned to all charts belonging to one trace. The background color can be changed by clicking the respective colored rectangle next to the trace file path in the Navigation Toolbar.

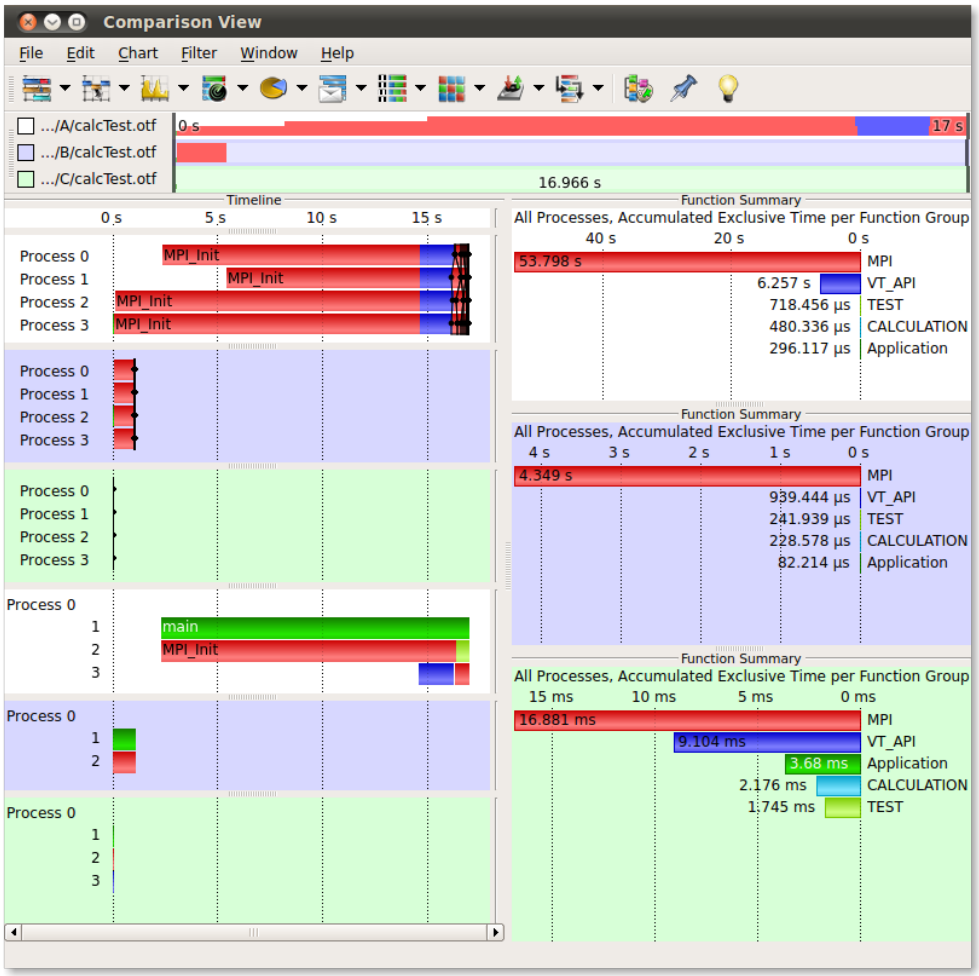


Figure 6.5: Comparison View with open charts

Figure 6.5 depicts a Comparison View with open Master Timeline, Process Timeline, and Function Summary charts.

All available charts work the same way as in the Trace View. Due to the fact that the Comparison View couples the zoom of all trace files, the charts can be used to directly compare performance characteristics between the traces.

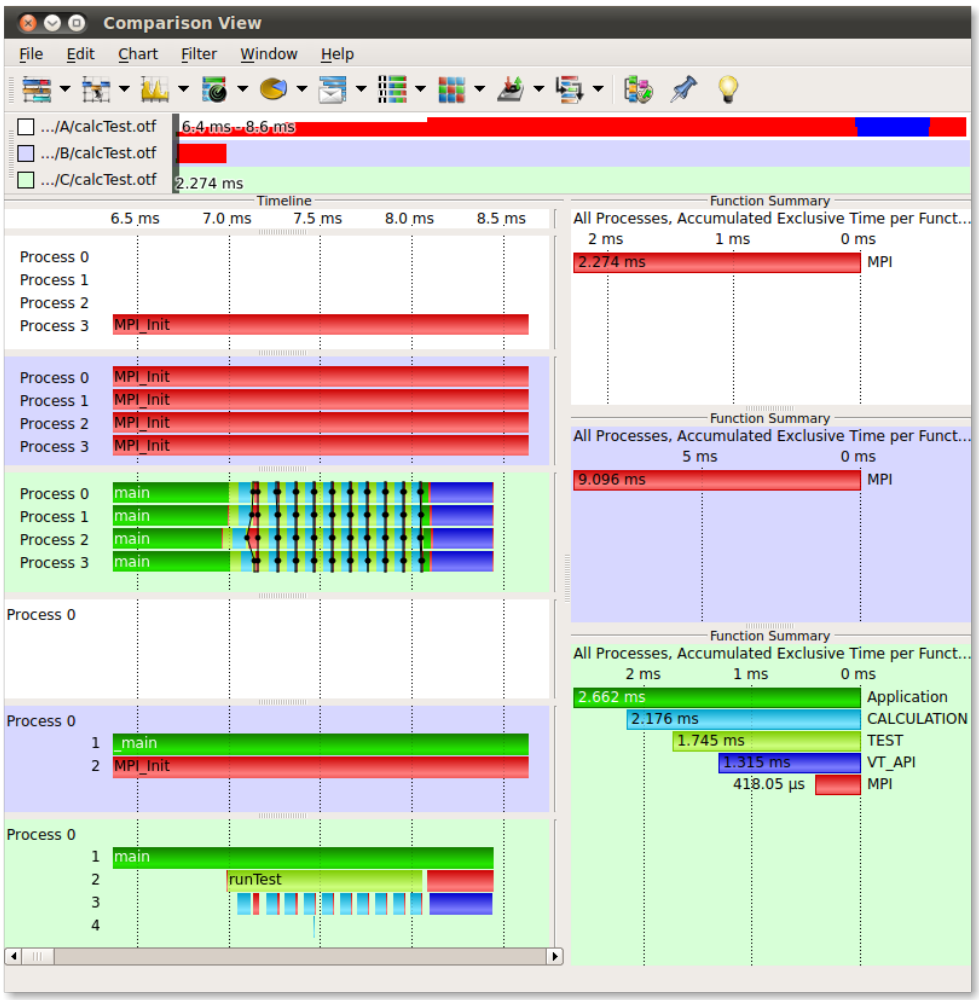


Figure 6.6: Zoom to compute iterations of trace C

As shown in Figure 6.5, trace A has the biggest duration time. The duration of trace C is so short that it is barely visible. Zooming into the compute iteration phase of trace C makes them visible but, due to the coupled zoom, also displays only the *MPI_Init* phase of trace A and B, see Figure 6.6. In order to compare the compute iterations between the traces they need to be aligned properly. This process is described in the next section.

6.3 Alignment of Multiple Trace Files

The Comparison View functionality to shift individual trace files in time allows to compare areas between traces that did not occur at the same time. For instance, in order to compare the compute iterations of the three example trace files these areas need to be aligned to each other. For the example traces this is required because the initialization of the application took different times on the three machines.

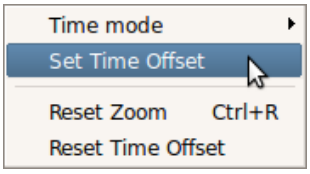


Figure 6.7: Context menu controlling the time offset

There are several ways to shift the trace files in time. One option is to use the context menu of the Navigation Toolbar. A right click on the toolbar reveals the menu as shown in Figure 6.7. The entry *Set Time Offset* allows to manually set the time offset for the respective trace file. The entry *Reset Time Offset* clears the offset.

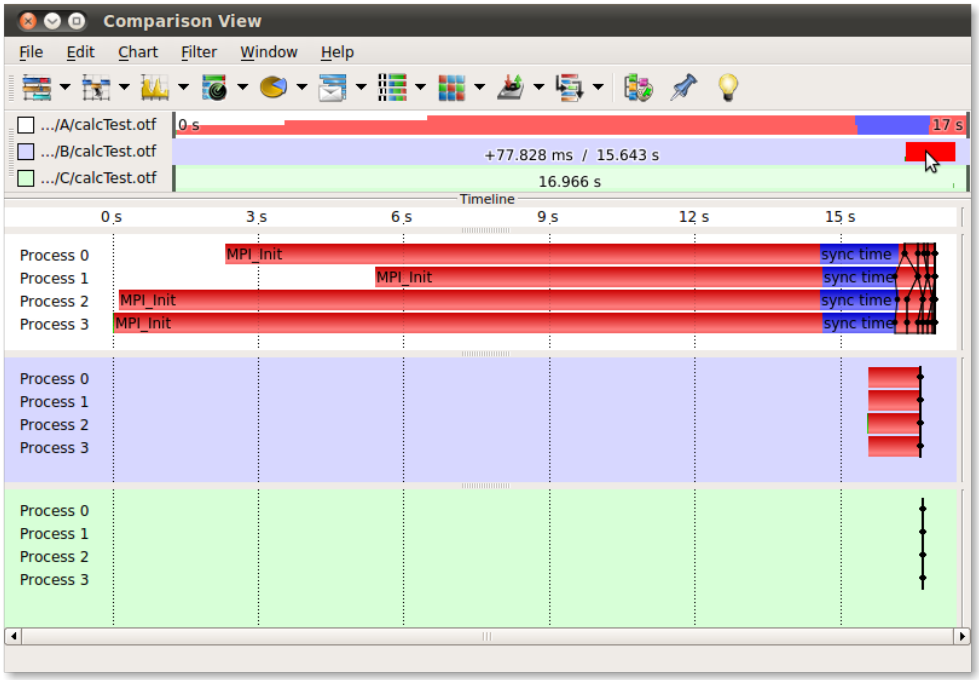


Figure 6.8: Alignment in the Navigation Toolbar

The easiest way to achieve a coarse alignment is to drag the trace file in the Navigation Toolbar. While holding the *Ctrl* (*Cmd* on Mac OS X) modifier key pressed the trace can

be dragged to the desired position with the left mouse button. In Figure 6.8 the compute iterations of all example trace files are coarsely aligned.

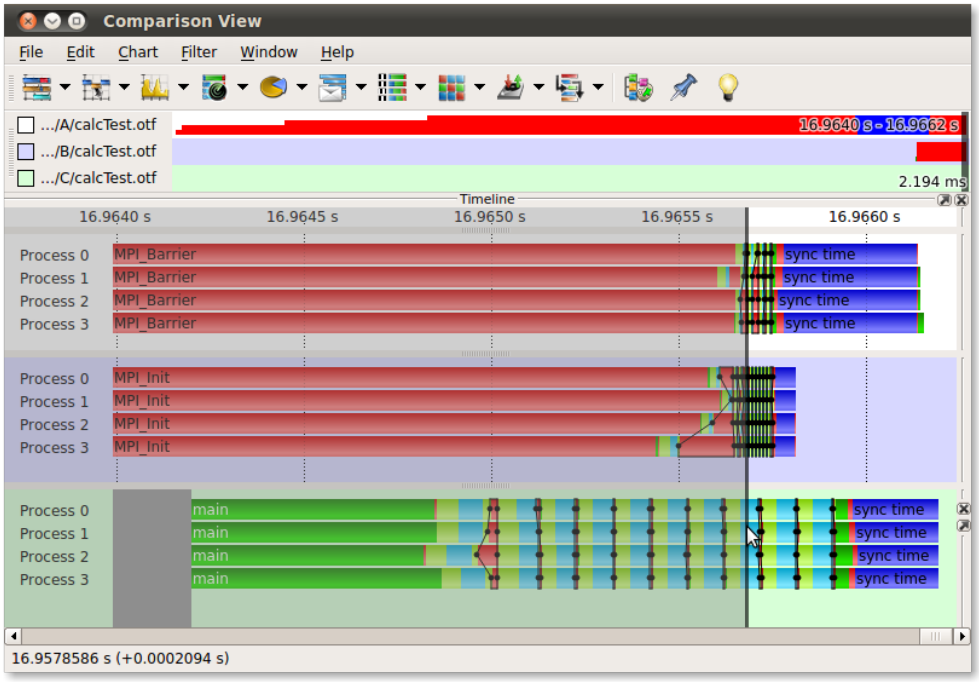


Figure 6.9: Alignment in the Master Timeline

After the coarse shifting a finer alignment can be achieved in the Master Timeline or Process Timeline charts. Therefore the user needs to zoom into the area to compare. Then, while keeping the *Ctrl* (*Cmd* on Mac OS X) modifier key pressed, the trace can be dragged with the left mouse button in the Master Timeline. Figure 6.9 depicts the process of dragging trace C to the compute iterations of trace A and B. As shown in the Figure 6.9, although the initialization of trace A took the longest, this machine was the fastest in computing the calculations.

6.4 Usage of Predefined Markers

Markers in traces point to particular places of interest in the trace data. These markers can be used to navigate in the trace files. For trace file comparison markers are interesting due to their potential to quickly locate places in large trace data sets. With the help of markers it is possible to find the same location in multiple trace files with just a few clicks.

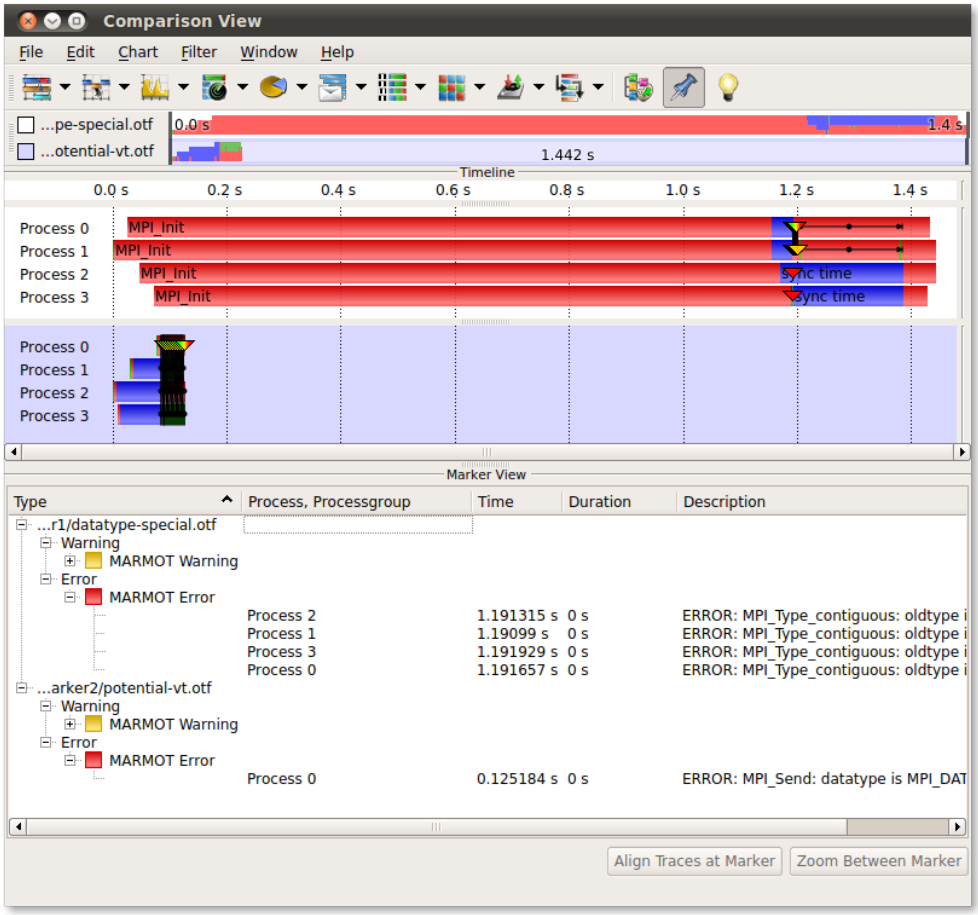


Figure 6.10: Open Marker View

First step in order to use markers is to open the Marker View. Figure 6.10 shows a Comparison View with an open Marker View. The markers of all open traces are shown combined in one Marker View. After a click on one marker in the Marker View the respective marker is highlighted in the Master Timeline and the Process Timeline.

Another way to navigate to a marker in the timeline charts is to use the Vampir zoom. If the user zoomed in the Master Timeline or the Process Timeline into the desired zooming level, then a click on a marker in the Marker View will shift the timeline zoom to the marker position. Thus, the selected marker appears in the center of the timeline chart, see Figure 6.11.

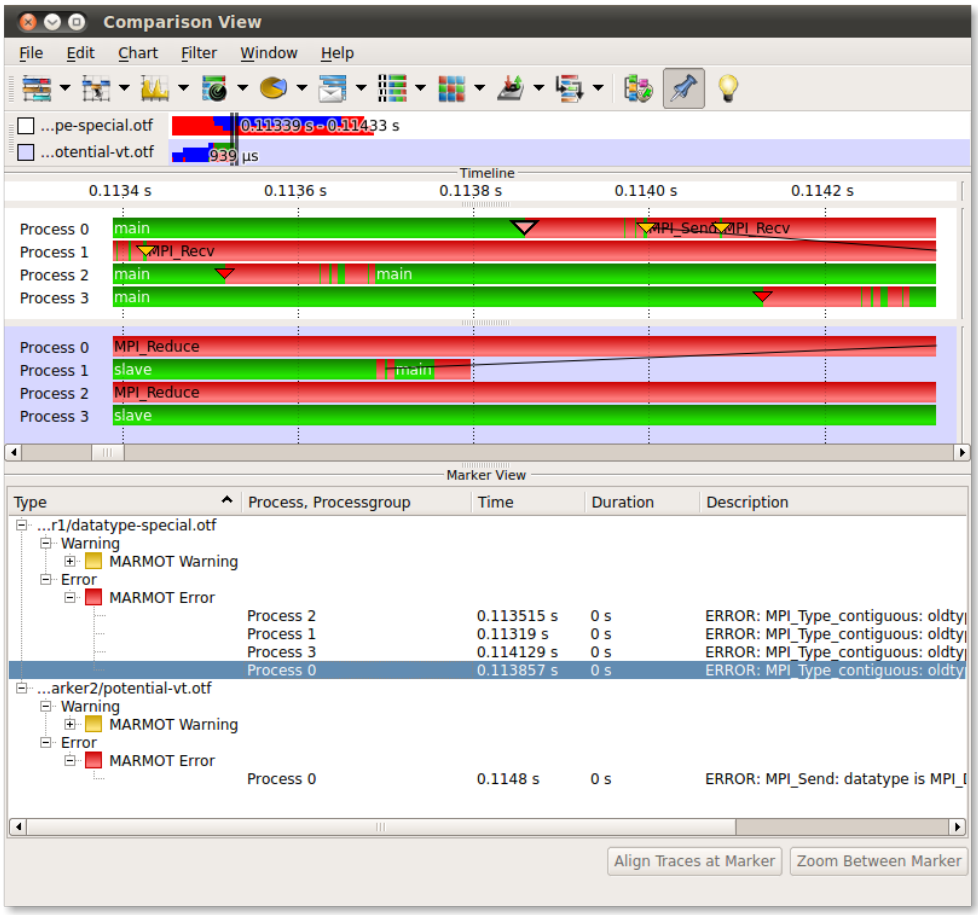


Figure 6.11: Jump to a marker in the Master Timeline

The Comparison View provides two additional ways of navigating with markers. If two markers of one trace are selected in the Marker View the button *Zoom Between Marker* sets the trace zoom to the according timestamps of the markers. If two markers of different traces are selected the button *Align Traces at Marker* adjusts the time offset between the respective traces. The selected markers are shown next to each other in the timeline charts, and consequently, both traces are aligned at the respective markers.

7 Customization

The appearance of the trace file and various other application settings can be altered in the preferences accessible via the main menu entry *File* → *Preferences*. Settings concerning the trace file itself, e.g. layout or function group colors are saved individually next to the trace file in a file with the ending *.vsettings*. This way it is possible to adjust the colors for individual trace files without interfering with others.

The options *Import Preferences* and *Export Preferences* provide the loading and saving of preferences of arbitrary trace files.

7.1 General Preferences

The *General* settings allow to change application and trace specific values.

Show time as decides whether the time format for the trace analysis is based on seconds or ticks.

With the *Automatically open context view* option disabled Vampir does not open the context view after the selection of an item, like a message or function.

Use color gradient in charts allows to switch off the color gradient used in the performance charts.

The next option allows to change the style and size of the font.

Show source code enables the internal source code viewer. This viewer shows the source code corresponding to selected locations in the trace file. In order to open a source file first click on the intended function in the *Master Timeline* and then on the source code path in the *Context View*. For the source code location to work properly, you need a trace file with source code location support. The path to the source file can be adjusted in the *Preferences* dialog. A limit for the size of the source file to be opened can be set, too.

In the *Analysis* section the number of analysis threads can be chosen. If this option is disabled, Vampir determines the number automatically by the number of cores, e.g. two analysis threads on a dual-core machine.

In the *Updates* section the user can decide if Vampir should check automatically for new versions.

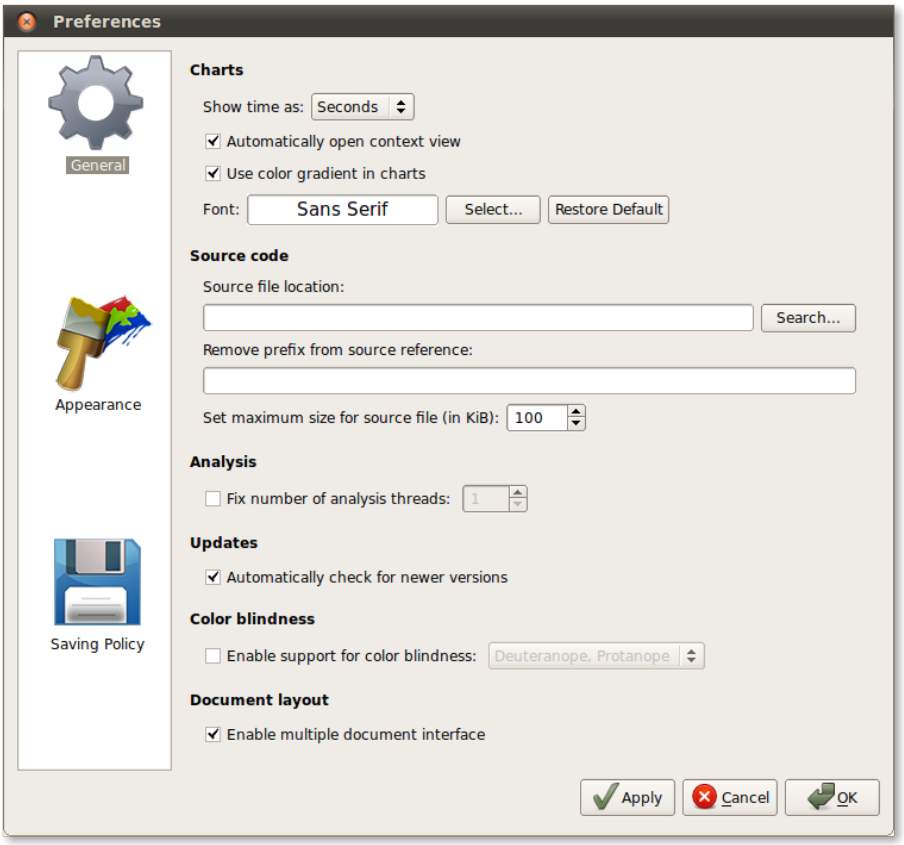


Figure 7.1: General Settings

Vampir also features a color blindness support mode.

On Linux systems there is also the *Document layout* option available. If this option is enabled all open *Trace View* windows need to stay in one main window. If it is disabled, the *Trace View* windows can be moved freely over the Desktop.

7.2 Appearance

In the *Appearance* settings of the *Preferences* dialog there are six different objects for which the color options can be changed. The functions/function groups, markers, counters, collectives, messages, and I/O events. Choose an entry and click on its color to make a modification. A color picker dialog opens where it is possible to adjust the color. For messages and collectives a change of the line width is also available.

In order to quickly find the desired item a search box is provided at the bottom of the dialog.

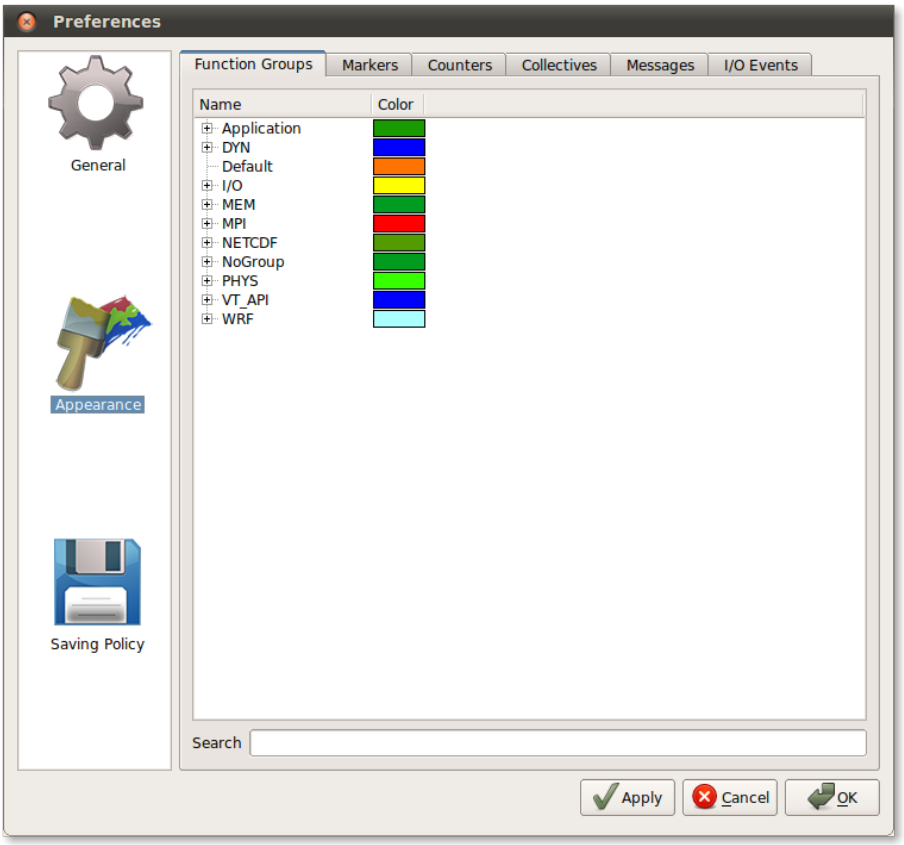


Figure 7.2: Appearance Settings

7.3 Saving Policy

Vampir detects whenever changes to the various settings are made. In the *Saving Policy* dialog it is possible to adjust the saving behavior of the different components to the own needs.

In the dialog *Saving Behavior* you tell Vampir what to do in the case of changed preferences. The user can choose the categories of settings, e.g., the layout, that should be affected by the selected behavior. Possible options are that the application automatically *Always* or *Never* saves changes. The default option is to have Vampir asking you whether to save or discard changes.

Usually the settings are stored in the folder of the trace file. If the user has no write access to it, it is possible to place them alternatively in the *Application Data Folder*. All such stored settings are listed in the tab *Locally Stored Preferences* with creation and modification date.

Note: On loading Vampir always favors settings in the *Application Data Folder*.

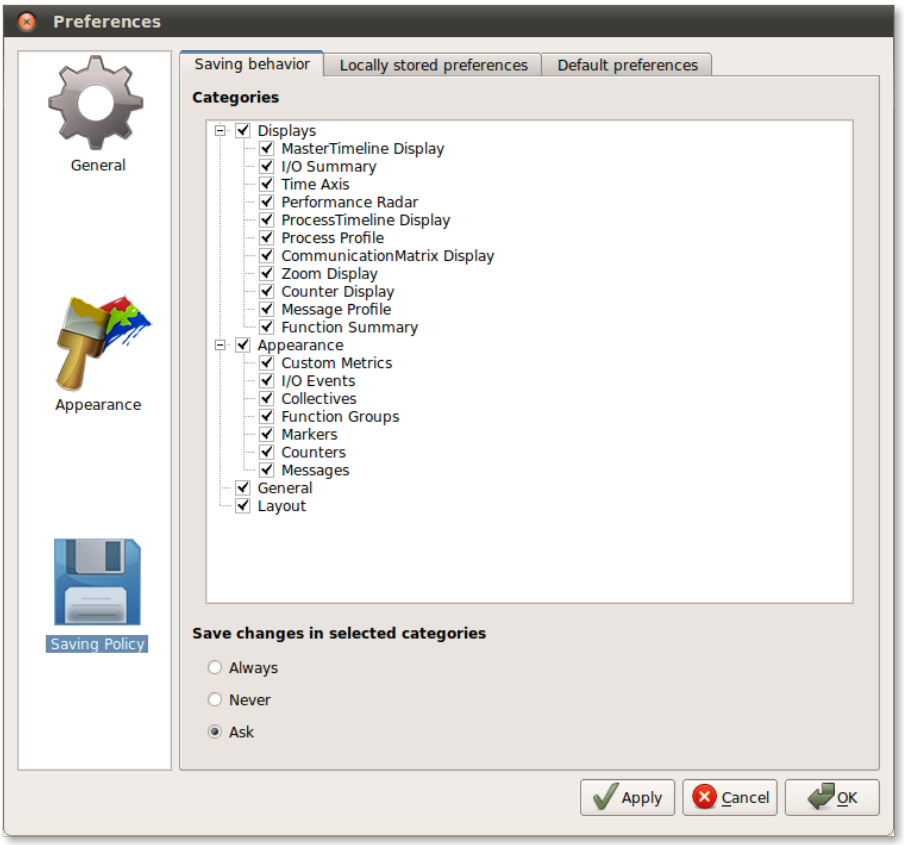


Figure 7.3: Saving Policy Settings

Default Preferences offers to save preferences of the current trace file as default settings. Then they are used for trace files without settings. Another option is to restore the default settings. Then the current preferences of the trace file are reverted.

8 A Use Case

This chapter explains by example how Vampir can be used to discover performance problems in your code and how to correct them.

8.1 Introduction

In many cases the Vampir suite has been successfully applied to identify performance bottlenecks and assist their correction. To show in which ways the provided toolset can be used to find performance problems in program code, one optimization process is illustrated in this chapter. The following example is a three-part optimization of a weather forecast model including simulation of cloud microphysics. Every run of the code has been performed on 100 cores with manual function instrumentation, MPI communication instrumentation, and recording of the number of L2 cache misses.

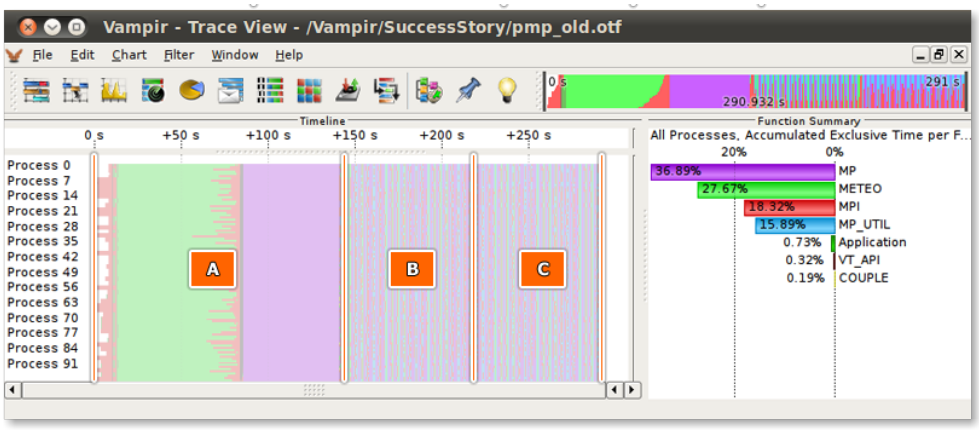


Figure 8.1: Master Timeline and Function Summary showing an overview of the program run

Getting a grasp of the program's overall behavior is a reasonable first step. In Figure 8.1 Vampir has been set up to provide such a high-level overview of the model's code. This layout can be achieved through two simple manipulations. Set up the Master Timeline to adjust the process bar height to fit the chart height. All 100 processes are now arranged into one view. Likewise, change the event category in the Function

Summary to show function groups. This way the many functions are condensed into fewer function groups.

One run of the instrumented program took 290 seconds to finish. The first half of the trace (Figure 8.1 A) is the initialization part. Processes get started and synced, input is read and distributed among these processes. The preparation of the cloud microphysics (function group: MP) is done here as well.

The second half is the iteration part, where the actual weather forecasting takes place. In a normal weather simulation this part would be much larger. But in order to keep the recorded trace data and the overhead introduced by tracing as small as possible only a few iterations have been recorded. This is sufficient since they are all doing the same work anyway. Therefore the simulation has been configured to only forecast the weather 20 seconds into the future. The iteration part consists of two “large” iterations (Figure 8.1 B and C), each calculating 10 seconds of forecast. Each of these in turn is partitioned into several “smaller” iterations.

For our observations we focus on only two of these small, inner iterations, since this is the part of the program where most of the time is spent. The initialization work does not increase with a higher forecast duration and would only take a relatively small amount of time in a real world run. The constant part at the beginning of each large iteration takes less than a tenth of the whole iteration time. Therefore, by far the most time is spent in the small iterations. Thus they are the most promising candidates for optimization.

All screenshots starting with Figure 8.2 are in a before-and-after fashion to point out what changed by applying the specific improvements.

8.2 Identified Problems and Solutions

8.2.1 Computational Imbalance

A varying size of work packages (thus varying processing time of this work) means waiting time in subsequent synchronization routines. This section points out two easy ways to recognize this problem.

Problem

As can be seen in Figure 8.2 each occurrence of the MICROPHYSICS-routine (purple color) starts at the same time on all processes inside one iteration, but takes between 1.7 and 1.3 seconds to finish. This imbalance leads to idle time in subsequent synchronization calls on the processes 1 to 4, because they have to wait for process 0 to finish its work (marked parts in Figure 8.2). This is wasted time which could be used for

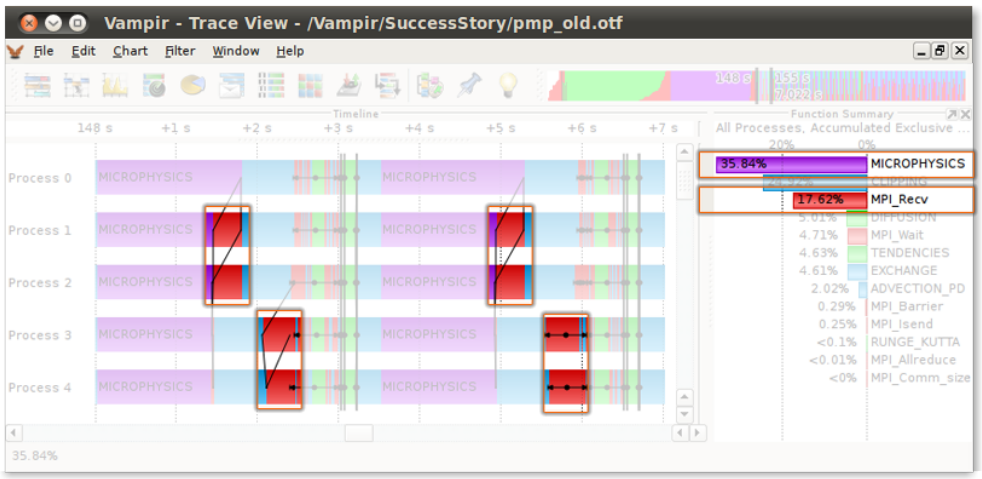


Figure 8.2: Before Tuning: Master Timeline and Function Summary identifying MICROPHYSICS (purple color) as predominant and unbalanced

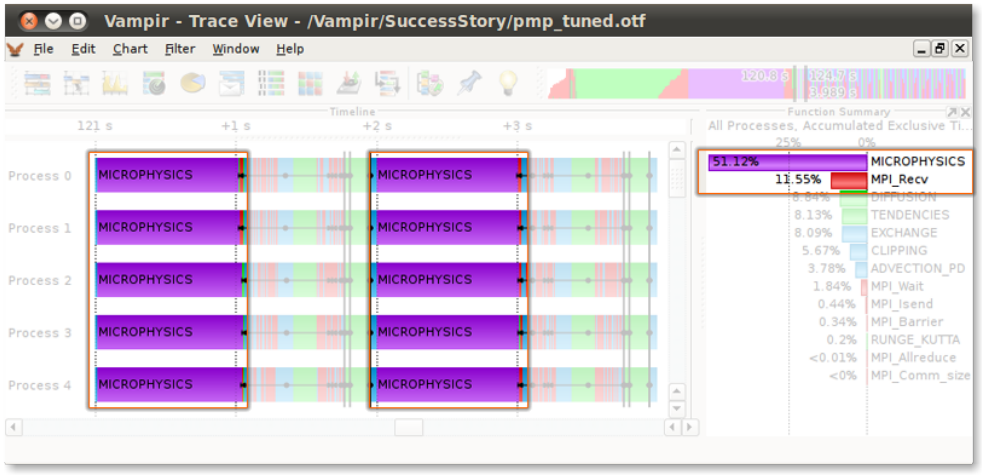


Figure 8.3: After Tuning: Timeline and Function Summary showing an improvement in communication behavior

computational work, if all MICROPHYSICS-calls would have the same duration. Another hint at this overhead in synchronization is the fact that the MPI receive routine uses 17.6% of the time of one iteration (Function Summary in Figure 8.2).

Solution

To even out this asymmetry the code which determines the size of the work packages for each process had to be changed. To achieve the desired effect an improved version of the domain decomposition has been implemented. Figure 8.3 shows that all occurrences of the MICROPHYSICS-routine are vertically aligned, thus balanced. Additionally the MPI receive routine calls are now clearly smaller than before. Comparing the Function Summary of Figure 8.2 and Figure 8.3 shows that the relative time spent in MPI receive has been decreased, and in turn the time spent inside MICROPHYSICS has been increased greatly. This means that we now spend more time computing and less time communicating, which is exactly what we want.

8.2.2 Serial Optimization

Inlining of frequently called functions and elimination of invariant calculations inside loops are two ways to improve the serial performance. This section shows how to detect candidate functions for serial optimization and suggests measures to speed them up.

Problem

All performance charts in Vampir show information of the time span currently selected in the timeline. Thus the most time-intensive routine of one iteration can be determined by zooming into one or more iterations and having a look at the Function Summary. The function with the largest bar takes up the most time. In this example (Figure 8.2) the MICROPHYSICS-routine can be identified as the most costly part of an iteration. Therefore it is a good candidate for gaining speedup through serial optimization techniques.

Solution

In order to get a fine-grained view of the MICROPHYSICS-routine's inner workings we had to trace the program using full function instrumentation. Only then it was possible to inspect and measure subroutines and subsubroutines of MICROPHYSICS. This way the most time consuming subroutines have been spotted, and could be analyzed for optimization potential.

The review showed that there were a couple of small functions which were called a lot. So we simply inlined them. With Vampir you can determine how often a functions is called by changing the metric of the Function Summary to the number of invocations.

The second inefficiency we discovered had been invariant calculations being done inside loops. So we just moved them in front of the respective loops.

Figure 8.3 sums up the tuning of the computational imbalance and the serial optimization. In the timeline you can see that the duration of the MICROPHYSICS-routine is now equal among all processes. Through serial optimization the duration has been decreased from about 1.5 to 1.0 second. A decrease in duration of about 33% is quite good given the simplicity of the changes done.

8.2.3 High Cache Miss Rate

The latency gap between cache and main memory is about a factor of 8. Therefore optimizing for cache usage is crucial for performance. If you don't access your data in a linear fashion as the cache expects, so called cache misses occur and the specific instructions have to suspend execution until the requested data arrives from main memory. A high cache miss rate therefore indicates that performance might be improved through reordering of the memory access pattern to match the cache layout of the platform.

Problem

As can be seen in the Counter Data Timeline (Figure 8.4) the CLIPPING-routine (light blue) causes a high amount of L2 cache misses. Also its duration is long enough to make it a candidate for inspection. What caused these inefficiencies in cache usage were nested loops, which accessed data in a very random, non-linear fashion. Data access can only profit from cache if subsequent read calls access data in the vicinity of the previously accessed data.

Solution

After reordering the nested loops to match the memory order, the tuned version of the CLIPPING-routine now needs only a fraction of the original time. (Figure 8.5)

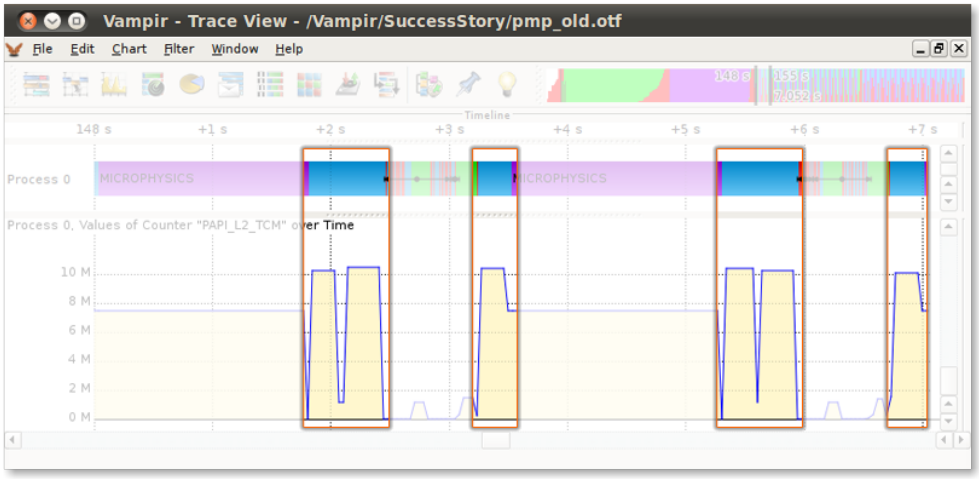


Figure 8.4: Before Tuning: Counter Data Timeline revealing a high amount of L2 cache misses inside the CLIPPING-routine (light blue)

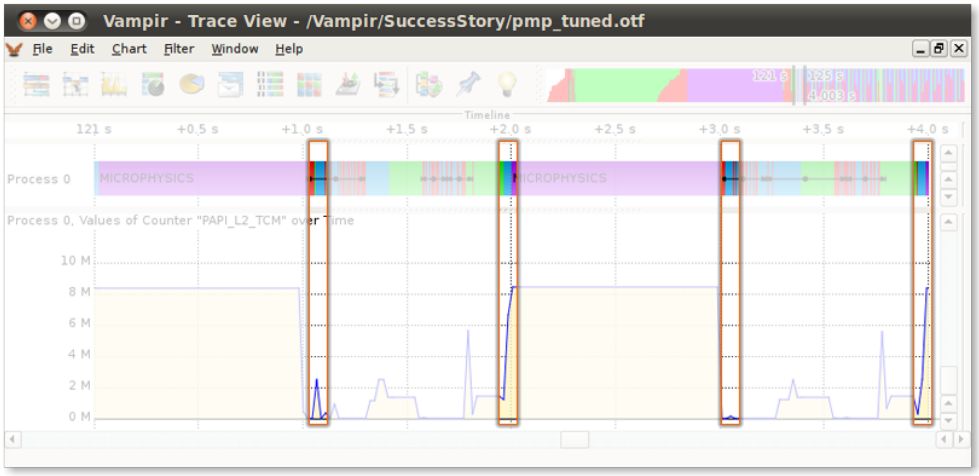


Figure 8.5: After Tuning: Visible improvement of the cache usage

8.3 Conclusion

By using the Vampir toolkit, three problems have been identified. As a consequence of addressing each problem, the duration of one iteration has been decreased from 3.5 seconds to 2.0 seconds.

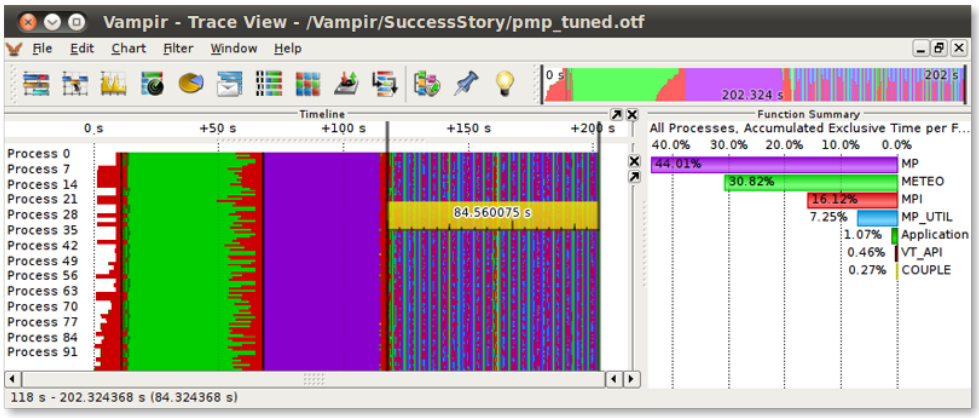


Figure 8.6: Overview showing a significant overall improvement

As is shown by the Ruler, see Section 4.1, in Figure 8.6 two large iterations now take 84 seconds to finish. Whereas at first (Figure 8.1) it took roughly 140 seconds, making a total speed gain of 40%.

This huge improvement has been achieved by using the insight into the program's runtime behavior, provided by the Vampir toolkit, to optimize the inefficient parts of the code.